ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Garbage Collecting Algorithm for Flash Disc

Αλγόριθμος επανάκτησης χώρου σε δίσκους στερεάς κατάστασης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μαρμαροκόπος Α. Γεώργιος

Επιβλέποντες Καθηγητές: Μποζάνης Παναγιώτης
Αναπληρωτής Καθηγητής

Τσομπανοπούλου Παναγιώτα
Επίκουρη Καθηγήτρια

Βόλος, Ιούλιος 2014

Στην οικογένειά μου και στους φίλους μου

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες καθηγητές μου, κ. Μποζάνη Παναγιώτη και κα. Τσομπανοπούλου Παναγιώτα για την καθοδήγησή τους κατά τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα κ. Φεύγα Αθανάσιο για την πολύτιμη και συνεχή βοήθειά του καθ' όλη τη διάρκεια υλοποίησης της παρούσας εργασίας.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στους φίλους μου για την αμέριστη υποστήριξη και την ανεκτίμητη βοήθεια που μου παρείχαν τόσο κατά την διάρκεια των σπουδών μου όσο και κατά την εκπόνηση της διπλωματικής εργασίας.

ΠΕΡΙΛΗΨΗ

Η μνήμη Flash σήμερα χρησιμοποιείται ευρέως σε πολλούς τομείς της τεχνολογίας καθώς και στις περισσότερες ηλεκτρονικές συσκευές λόγω του μεγέθους και των ταχυτήτων που προσφέρει έναντι των μαγνητικών σκληρών δίσκων. Τα χαρακτηριστικά που διέπουν μια μνήμη Flash και συγκεκριμένα την NAND μνήμη Flash καθιστούν αναγκαία την ύπαρξη ορισμένων λειτουργιών όπως αυτή του Garbage Collection.

Στην παρούσα διπλωματική παρουσιάζεται ο τρόπος λειτουργίας του Garbage Collection στη μνήμη Flash καθώς επίσης αναλύονται συγκεκριμένοι αλγόριθμοι που προσπαθούν να υλοποιήσουν τη παραπάνω λειτουργία με διαφορετικό τρόπο παρουσιάζοντας τα χαρακτηριστικά και τις διαφορές τους.

Στο πρακτικό μέρος της εργασίας, αναπτύχθηκε ένας εικονικός προσομοιωτής Garbage Collection μνήμης Flash σε προγραμματιστική γλώσσα Java για την περαιτέρω ανάλυση και την οπτική παρουσίαση του κάθε αλγορίθμου.

ABSTRACT

Nowadays, Flash memory is widely used in many fields of technology as well as in most electronic devices due to its size and its speed which offers over the magnetic hard discs. The attributes that govern a Flash memory and especially the NAND Flash memory necessitate the existence of certain functions such as the Garbage Collection.

In this thesis we present the Garbage Collection function in Flash memory and analyze specific algorithms that try to implement the above function in a different manner by presenting their features and their differences.

In the practical section of this thesis, is developed a virtual Garbage Collection simulator for Flash memory in Java programming language for further analysis and visual presentation of each algorithm.

Περιεχόμενα

Πίνακας εικόνων

Πίνακας Διαγραμμάτων

1. Εισαγωγή

Στην παρούσα διπλωματική παρουσιάζεται γενικά η μνήμη flash καθώς περιγράφονται η αρχιτεκτονική της, οι βασικές κατηγορίες της όπως επίσης και οι διαφορές των συγκεκριμένων κατηγοριών.

Στο κεφάλαιο 2 αναλύονται τα χαρακτηριστικά ειδικά της NAND μνήμης flash τα οποία είναι και ο λόγος ύπαρξης των Garbage Collection και Wear Leveling.

Παράλληλα στο κεφάλαιο 3 παρουσιάζονται οι μηχανισμοί της συλλογής απορριμμάτων και εξισορρόπησης φθοράς περιγράφοντας τον τρόπο λειτουργίας τους μέσα σε μία μνήμη flash. Επιπρόσθετα αναλύονται ξεχωριστά διάφοροι αλγόριθμοι που δημιουργήθηκαν με σκοπό την βελτιστοποίηση των μηχανισμών αυτών και κυρίως του Garbage Collection.

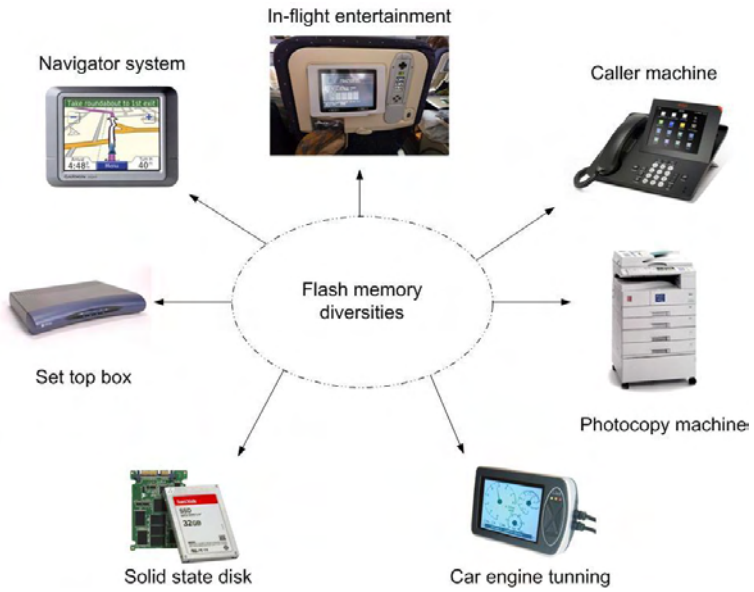
Επιπλέον, στο κεφάλαιο 4 περιγράφεται ο τρόπος ανάπτυξης ενός εικονικού προσομοιωτή ο οποίος αποτέλεσε το πρακτικό μέρος της εργασίας. Πιο αναλυτικά, περιγράφονται βασικές συναρτήσεις που υλοποιήθηκαν για τον κάθε αλγόριθμο, γενικές συναρτήσεις που είναι χρήσιμες για την σωστή λειτουργία του όπως επίσης η τελική μορφή του και οι λειτουργίες του.

Τέλος παρουσιάζονται τα αποτελέσματα του κάθε αλγόριθμου σε διαγράμματα από τα οποία μπορούμε να συγκρίνουμε την αποτελεσματικότητά τους.

2. Flash Memory

2.1. Εισαγωγή

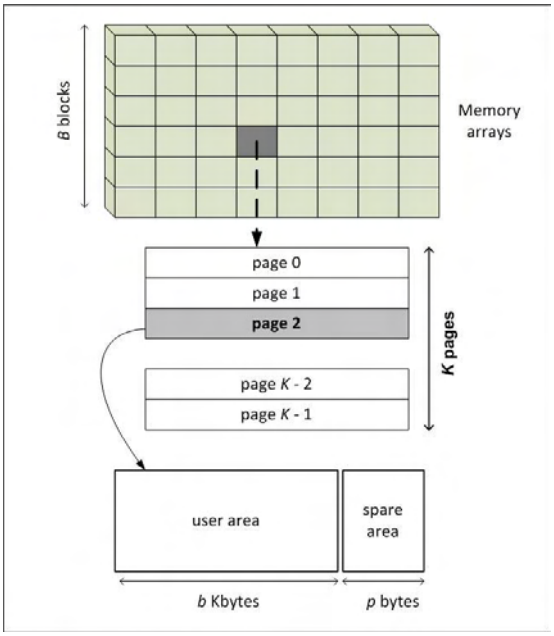
Η μνήμη flash είναι μία μη ευμετάβλητη συσκευή αποθήκευσης που μπορεί να διατηρήσει το περιεχόμενό της όταν η συσκευή είναι απενεργοποιημένη. Γενικότερα, χαρακτηρίζεται ως ένας EEPROM (*Electronically Erasable Programmable Read-Only Memory*) τύπος μνήμης που προσφέρει πολλά και εξαιρετικά χαρακτηριστικά όπως λιγότερο θόρυβο, solid-state αξιοπιστία, ελάχιστη απαίτηση κατανάλωσης ενέργειας (energy-efficient), μικρότερο μέγεθος, μικρό βάρος και αυξημένη αντοχή σε δονήσεις (shock resistant). Προσφέρει επίσης γρήγορους χρόνους πρόσβασης για ανάγνωση, χωρίς ωστόσο να είναι ταχύτερη από την κύρια μνήμη DRAM ενός συμβατικού υπολογιστή. Έτσι η μνήμη flash λειτουργεί ως μία λεπτή και συμπαγής συσκευή αποθήκευσης με αποτέλεσμα να αποτελεί τον ιδανικό τύπο μνήμης στις περισσότερες φορητές ηλεκτρονικές συσκευές όπως κινητά τηλέφωνα, ψηφιακές φωτογραφικές μηχανές, προσωπικούς ψηφιακούς βοηθούς (PDA), φορητές συσκευές αναπαραγωγής πολυμέσων (PMPs), δέκτες παγκόσμιου συστήματος εντοπισμού θέσης (GPS) και σε πολλές άλλες. Τέλος η ζήτηση της μνήμης flash είναι τόσο μεγάλη πού χρησιμοποιείται ευρέως και σε άλλους τομείς της τεχνολογίας. Για παράδειγμα, όπως απεικονίζεται στην [Εικόνα 1], η μνήμη flash χρησιμοποιείται εκτενώς σε ενσωματωμένα συστήματα, σε αρκετές έξυπνες και καινοτόμες εφαρμογές όπως οικιακές συσκευές, συσκευές τηλεπικοινωνίας, εφαρμογές υπολογιστών, στην αυτοκινητοβιομηχανία και σε μηχανήματα υψηλής τεχνολογίας.



Εικόνα 1: Η χρήση της μνήμης Flash σε διάφορες συσκευές (http://www.intechopen.com/books/flash-memories/block-cleaning-process-in-flash-memory)

2.2. Αρχιτεκτονική της μνήμης Flash

Η μνήμη flash όπως παρουσιάζεται στην [Εικόνα 2] είναι μία συσκευή αποθήκευσης βασισμένη σε μπλοκ και σελίδες όπου κάθε σελίδα χρησιμοποιείται για την αποθήκευση δεδομένων και μια ομάδα από σελίδες αποτελούν με την σειρά τους ένα μπλοκ. Η κάθε σελίδα έχει μια περιοχή με κύρια δεδομένα (*main data area*) και μια ελεύθερη περιοχή *(spare area*). Η περιοχή των δεδομένων χρησιμοποιείται για την αποθήκευση των πραγματικών δεδομένων, ενώ η πρόσθετη έκταση για την αποθήκευση των υποβοηθητικών στοιχείων που είναι χρήσιμα για τα πραγματικά δεδομένα (όπως η αναγνώριση των προβληματικών μπλοκ (bad blocks), οι δομές δεδομένων των σελίδων και των μπλοκ, ο κώδικας διόρθωσης σφαλμάτων (Error Correction Code), κλπ.). Σύμφωνα με τα σημερινά δεδομένα, το μέγεθος της σελίδας είναι σταθερό και κυμαίνεται από 512 B έως 8 KB, ενώ το μέγεθος του μπλοκ είναι μεταξύ 4 KB έως 128 KB. Η [Εικόνα 3] δείχνει τα χαρακτηριστικά μιας μνήμης flash 4 GB.



Εικόνα 2: Η μνήμη Flash βασισμένη σε μπλοκ και σελίδες
(http://www.intechopen.com/books/flash-memories/block-cleaning-process-in-flash-memory)

Symbol	Description	Value
-	Power consumption	2.70 – 3.60 V
-	Dimension (h x w x d) mm	12 x 20 x 1.2
R_t	Data read from a page into the data register	25 μs
W_t	Data write from a data register into memory page	200 μs
E_t	Block erasure access time	1.5 ms
S_t	Sequential access to the data register (bus data)	100 μs
V	Number of chips per device	2
B	Number of blocks per chip	8192
K	Number of pages per block	64
p	Page size	4 KB
b	Block size	256 KB
r_{data}	Data register	4 KB
D	Capacity per chip	2 GB
u_i	Block i utilization, $0 \le i \le B-1$	[0, 1]
-	Program/Erase cycles	1,000,000

Εικόνα 3: Τα χαρακτηριστικά μιας μνήμης flash 4 GB
(http://www.intechopen.com/books/flash-memories/block-cleaning-process-in-flash-memory)

2.3. Κατηγορίες της μνήμης Flash

Η μνήμη Flash διακρίνεται σε δύο κατηγορίες: την NOR μνήμη Flash και την NAND μνήμη Flash. Η ονομασία τους οφείλεται στο τύπο των λογικών πυλών που χρησιμοποιούνται για κάθε κελί της μνήμης (όπου λογικές πύλες είναι μπλοκ από ψηφιακά κυκλώματα). Τις μνήμες Flash τις γνωρίσαμε το 1988 όταν η εταιρεία INTEL εισήγαγε την NOR μνήμη Flash και το 1989 όταν η εταιρεία TOSHIBA εισήγαγε την NAND μνήμη Flash.

2.3.1. NAND Μνήμη Flash

Στον τύπο της NAND μνήμης Flash, το chip διακρίνεται από ένα συγκεκριμένο αριθμό μπλοκ (*block*), με το κάθε μπλοκ να έχει ένα καθορισμένο αριθμό σελίδων (*pages*). Κάθε μπλοκ αυτού του τύπου μνήμης, μπορεί να σβηστεί (*erased*) μέχρι 1.000.000 φορές ,αλλά αν ξεπεραστεί το νούμερο αυτό τότε δημιουργείται φθορά του μπλοκ (*worn out*) με αποτέλεσμα να υπάρχουν λάθη στην λειτουργία της εγγραφής.

Υπάρχουν διάφοροι τύποι NAND μνήμης Flash. Ο πρώτος λέγεται Μονού-Επιπέδου Κελιού (Single-Level Cell – SLC) μνήμη NAND και έχει οργανωθεί με τέτοιο τρόπο ώστε το κάθε μπλοκ να έχει 32 σελίδες, με το μέγεθος κάθε σελίδας να είναι (512+16) bytes, που συμπεριλαμβάνει και τα 16 bytes της ελεύθερης περιοχής. Ο δεύτερος, πιο σύγχρονος τύπος λέγεται ’’Μεγάλου μπλοκ SLC NAND’’ (large block SLC NAND) μνήμη NAND. Σε σύγκριση με τον προηγούμενο τύπο, διαθέτει μεγαλύτερη χωρητικότητα, με διπλάσιο αριθμό σελίδων σε κάθε μπλοκ και τετραπλάσιο μέγεθος της κάθε σελίδας από αυτόν. Ο τρίτος τύπος είναι μεταγενέστερος και ονομάζεται ως Πολλαπλού-Επιπέδου Κελιού (Multiple-Level Cell MLC NAND) μνήμη NAND. Στον συγκεκριμένο τύπο το κάθε τρανζίστορ, μπορεί να υπάρχει σε μία από τις 4 διαφορετικές καταστάσεις, η δυνατότητα κωδικοποίησης δεδομένων και η δυνατότητα αποθήκευσης δύο bit ανά κελί μνήμης (memory cell). Με αυτόν τον τρόπο έχουμε διπλασιασμό της χωρητικότητας της NAND μνήμης Flash, ώστε κάθε σελίδα να αποθηκεύει 4096 bytes δεδομένων, ενώ το μέγεθος της ελεύθερης περιοχής της να είναι 128 bytes. Ο δε αριθμός των σελίδων που περιέχονται σε κάθε μπλοκ ανέρχεται στις 128.

2.3.2. NOR Μνήμη Flash

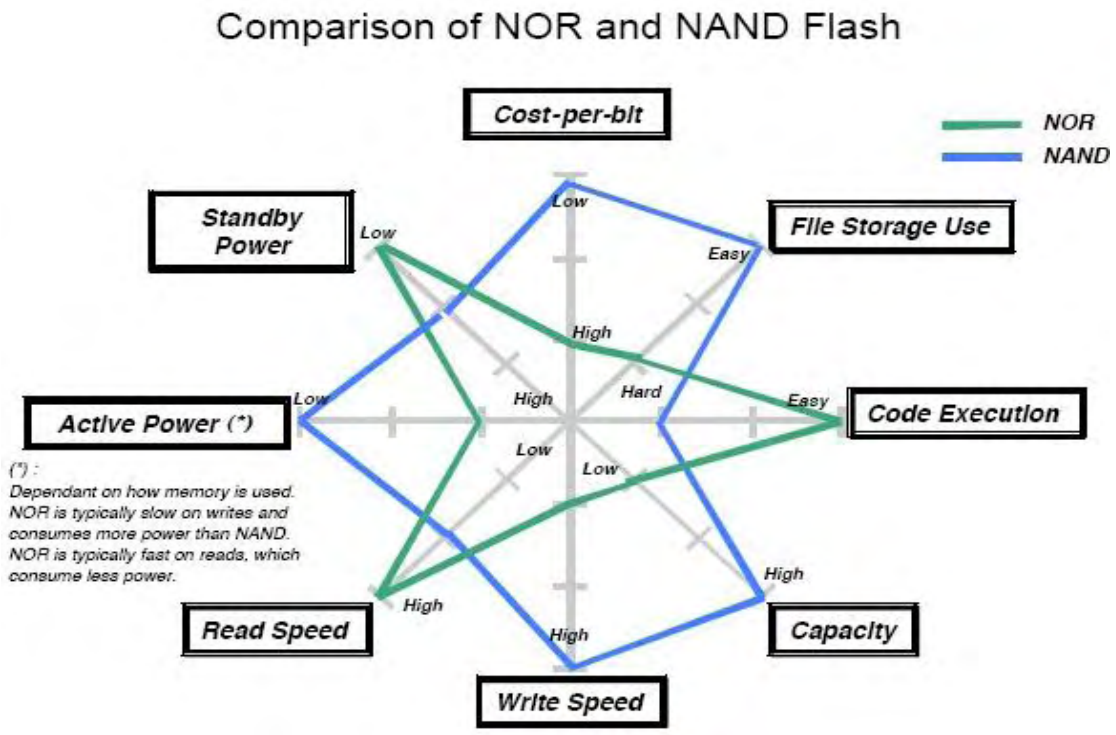
Η NOR μνήμη Flash εμφανίστηκε πρώτη από τους δύο τύπους και παρέχει τυχαία προσπέλαση (random access). Στις μνήμες αυτές τα περιεχόμενα μπορούν να προσπελαστούν ή να προγραμματιστούν ανά λέξη ,ενώ η διαγραφή (όλα τα bits τίθενται σε '1') λαμβάνει χώρα ανά μπλοκ διευθύνσεων. Η Κεντρική Μονάδα Επεξεργασίας (CPU) δημιουργεί διευθύνσεις σε επίπεδο byte και αν χρειαστεί μπορεί να χρησιμοποιηθεί και σαν κύρια μνήμη RAM. Ως επί το πλείστον χρησιμοποιείται για να αποθηκεύονται στατικά δεδομένα, αφού οι χρόνοι εγγραφής είναι υψηλοί.

Στα υπέρ της NOR μνήμης Flash είναι ότι η εγγραφή των δεδομένων γίνεται σε επίπεδο byte ενώ στην NAND μνήμη Flash η εγγραφή γίνεται σε επίπεδο σελίδας. Επιπλέον η NOR μνήμη Flash έχει πιο γρήγορους χρόνους προσπέλασης δεδομένων (200ns) σε σχέση με την NAND μνήμη Flash (50-80μs), ενώ αντιθέτως χάνει σε πυκνότητα (density) και αποδοτικότητα ισχύος (power efficiency).

2.3.3. Διαφορές των κατηγοριών της μνήμης Flash

Μεταξύ αυτών των δύο τύπων μνημών υπάρχουν αρκετές διαφορές. Η NOR μνήμη Flash ,είναι ταχύτερη αλλά κοστίζει περισσότερο, ενώ είναι πιο αργή στην λειτουργία της διαγραφής και της εγγραφής δεδομένων σε συγκεκριμένες διευθύνσεις μνήμης , με συνηθέστερη εφαρμογή της στην κινητή τηλεφωνία. Η NAND μνήμη Flash έχει σαν πλεονέκτημα την μεγαλύτερη χωρητικότητα αποθήκευσης έναντι της NOR μνήμης Flash.

Και στις δύο αυτές κατηγορίες μνημών, στις λειτουργίες εγγραφής η τιμή των bits επανέρχεται στην αρχική τιμή 0 (*clear bits* – αλλάζουν την τιμή από 1 σε 0). Εν συνεχεία για να αλλάξουν τιμή τα bits (*set bits* – αλλαγή της τιμής από 0 σε 1) πρέπει να σβηστεί ένα μεγάλο μέρος της μνήμης. Το τμήμα της μνήμης που σβήνεται χαρακτηρίζεται ως Μονάδα σβησίματος (*erase unit*) και είναι συγκεκριμένου και καθορισμένου μεγέθους.



Εικόνα 4: Σύγκριση των NOR και NAND
(http://embeddeddomain.blogspot.gr/2011/04/nor-vs-nand-flash.html)

Για να είναι λοιπόν η μνήμη Flash όσο το δυνατόν αποδοτικότερη, χρησιμοποιούνται ειδικές δομές δεδομένων και αλγόριθμοι. Η χρησιμοποίησή τους (αλγόριθμοι και ειδικές δομές δεδομένων) δημιουργεί νέες και διαφορετικές θέσεις στην ανανέωση των δεδομένων (not-in-place updates of data), ενώ φροντίζουν να κρατούν ισορροπία στην φθορά των μπλοκ της μνήμης και λιγοστεύουν κατά πολύ τις διαγραφές.

2.4. Χαρακτηριστικά της NAND μνήμης Flash

2.4.1. Out-place & in place σύστημα ενημέρωσης:

Όπως αναφέρεται παραπάνω, τα δεδομένα σε μια μνήμη flash είναι αποθηκευμένα σε μπλοκ. Όταν κάποια από αυτά τα δεδομένα ενημερωθούν τότε υπάρχουν δύο τρόποι για να αποθηκευτούν ξανά.

Ο πρώτος λέγεται **in-place scheme** ενώ ο δεύτερος **out-place scheme**. Κατά τον πρώτο τρόπο λόγω του ότι η μνήμη flash είναι ένας EEPROM (*Electronically Erasable Programmable Read-Only Memory*) τύπος μνήμης τα νέα ‘’ενημερωμένα’’

δεδομένα αποθηκεύονται στο μπλοκ όπου υπήρχαν τα ‘’παλιά’’, αφού όμως πρώτα έχουν σβηστεί/καθαριστεί. Όμως η διαγραφή ενός μπλοκ στην μνήμη flash είναι χρονοβόρα με αποτέλεσμα να υποβαθμίζεται η I/O απόδοση.

Έτσι η ενημέρωση των δεδομένων σε μία μνήμη flash γίνεται μέσω του out-place scheme και όχι μέσω ενός in-place scheme. Σε αυτή την περίπτωση τα νέα ‘’ενημερωμένα’’ δεδομένα αποθηκεύονται σε μια καινούρια τοποθεσία μέσα στην μνήμη, ενώ τα παλιά χαρακτηρίζονται ως μη έγκυρα και δεν χρησιμοποιούνται μέχρι να καθαριστούν. Ο κύριος σκοπός του out-place scheme είναι να αποφευχθεί η διαγραφή μπλοκ κατά τη διάρκεια κάθε διαδικασίας ενημέρωσης.

Λόγω του out-place scheme μία σελίδα της μνήμης flash μπορεί να βρίσκεται σε μία από τις τρείς καταστάσεις: **ελεύθερη (free), έγκυρη (valid)** και **μη έγκυρη (invalid).** Σε ελεύθερη (free) κατάσταση βρίσκεται όταν η σελίδα δεν περιέχει δεδομένα και είναι διαθέσιμη προς αποθήκευση (γράφοντας σε αυτήν νέα ή ενημερωμένα δεδομένα). Σε έγκυρη (valid) κατάσταση είναι η σελίδα που περιέχει την τρέχουσα έκδοση των δεδομένων ενώ η μη έγκυρη (invalid) κατάσταση παραπέμπει σε μία σελίδα που περιέχει επαρχαιωμένα δεδομένα που χαρακτηρίζονται ως ‘’απορρίμματα’’. Επιπλέον, όσον αφορά τα μπλοκ της μνήμης η κατάστασή τους χαρακτηρίζεται ως active ή inactive ανάλογα με τις σελίδες που εκείνα περιέχουν.

2.4.2. Λειτουργίες με διαφορετικούς χρόνους και μονάδες (page, block) προσπέλασης:

Υπάρχουν τρεις βασικές λειτουργίες πρόσβασης σε μία μνήμη flash η **ανάγνωση(read)**, η **εγγραφή(write)** και το **σβήσιμο(erase).** Η NAND –flash μνήμη, λόγω της οργάνωσής της, αδυνατεί να διαβάσει ή να εγγράψει κάθε κελί ξεχωριστά. Η μνήμη ομαδοποιείται και είναι προσβάσιμη σε πολύ συγκεκριμένες διαδικασίες οι οποίες είναι ζωτικής σημασίας για την βελτιστοποίηση των δομών δεδομένων και την κατανόηση της συμπεριφοράς της. Παρακάτω αναλύονται οι βασικές λειτουργίες της NAND-flash μνήμης.

2.4.2.1. *Ανάγνωση (Read)*

Η ανάγνωση σε μία μνήμη flash γίνεται στο μέγεθος μιας σελίδας.

Σε μία μνήμη flash δεν είναι δυνατό να διαβαστεί κάτι μικρότερο από το μέγεθος μιας σελίδας και αυτό είναι ένα από τα χαρακτηριστικά της. Για παράδειγμα μπορεί να ζητηθεί από την μνήμη ένα byte προς ανάγνωση. Σε αυτή την περίπτωση θα δοθεί μια ολόκληρη σελίδα (4kB) , αναγκάζοντας να αναγνωστούν πολλά περισσότερα δεδομένα από ότι χρειάζεται στην πραγματικότητα.

2.4.2.2. *Εγγραφή (Write)*

Η εγγραφή σε μία μνήμη flash γίνεται σε μέγεθος μιας σελίδας.

Όπως ακριβώς γίνεται και στην ανάγνωση, για να γίνει εγγραφή ενός byte χρειάζεται να δοθεί από την μνήμη μία ολόκληρη σελίδα με αποτέλεσμα να εγγράφονται περισσότερα "δεδομένα" από ότι είναι απαραίτητο στην πραγματικότητα. Αυτό το γεγονός είναι γνωστό ως "Write Amplification".

Παρατήρηση: Μία σελίδα δεν μπορεί να ξαναεγγραφεί χωρίς πρώτα να σβηστεί.

Ένα επιπλέον χαρακτηριστικό της εγγραφής σε μια μνήμη flash είναι ότι μία σελίδα μπορεί να γραφτεί μόνο αν βρίσκεται σε ελεύθερη κατάσταση. Έτσι όταν κάποια δεδομένα αλλαχτούν, τα περιεχόμενα της σελίδας ενημερώνονται και η νέα έκδοση των δεδομένων αυτών αποθηκεύεται σε μία νέα ελεύθερη σελίδα διαφορετική από εκείνη που περιείχε τα αρχικά δεδομένα. Μόλις γίνει η ενημέρωση η αρχική σελίδα επισημαίνεται ως ‘’invalid’’ και θα παραμείνει ως έχει μέχρι να διαγραφεί.

2.4.2.3. *Σβήσιμο (Erase)*

Το σβήσιμο σε μία μνήμη flash γίνεται στο μέγεθος ενός μπλοκ.

Όπως ήδη έχει αναφερθεί παραπάνω μία σελίδα δεν μπορεί να ξαναεγγραφεί και από την στιγμή που γίνεται μη έγκυρη ο μόνος τρόπος για να μπορέσει να χρησιμοποιηθεί ξανά ως ελεύθερη είναι να διαγραφούν τα δεδομένα της. Ωστόσο σε μία μνήμη flash δεν είναι δυνατό να διαγραφεί μόνο μια σελίδα αλλά ένα ολόκληρο μπλοκ. Από την σκοπιά όμως των χρηστών οι μόνες εντολές που μπορούν να

χρησιμοποιήσουν κατά την πρόσβαση τους στα δεδομένα είναι αυτές της ανάγνωσης και της εγγραφής. Η εντολή του σβησίματος ενεργοποιείται αυτόματα κατά τη διαδικασία της συλλογής απορριμμάτων (garbage collection) όταν χρειάζεται η μνήμη να διεκδικήσει εκ νέου τις μη έγκυρες σελίδες για να δημιουργηθεί ελεύθερος χώρος.

Τέλος, η κάθε μία από τις λειτουργίες αυτές, έχει διαφορετικό χρόνο προσπέλασης στην μνήμη. Η εγγραφή λαμβάνει μία τάξη μεγέθους μεγαλύτερη από την λειτουργία ανάγνωσης ενώ από την άλλη το σβήσιμο στην μνήμη απαιτεί τον περισσότερο χρόνο σε σχέση με τις υπόλοιπες. Κατά την ανάγνωση ανακτώνται έγκυρα δεδομένα (valid data) από μία έγκυρη σελίδα, ενώ κατά την εγγραφή, δεδομένα (είτε νέα είτε ενημερωμένα) αποθηκεύονται σε ελεύθερες σελίδες. Αντιθέτως κατά την λειτουργία του σβησίματος χρησιμοποιείται ένα ενεργό ή ανενεργό μπλοκ που μπορεί να περιέχει μη έγκυρες (invalid) και ελεύθερες σελίδες.

2.4.3. Μαζικός καθαρισμός και μπλοκ με περιορισμένο κύκλο ζωής:

Η διαδικασία καθαρισμού είναι απαραίτητη στη μνήμη flash λόγω του **out-place scheme**. Το κάθε μπλοκ που σβήνεται μπορεί να περιέχει και κάποια έγκυρα δεδομένα και γι' αυτόν τον λόγο πρέπει, πριν την έναρξη του σβησίματος, αυτά τα δεδομένα να μεταφερθούν σε άλλα μπλοκ με ελεύθερο χώρο. Ωστόσο κάθε μπλοκ έχει έναν περιορισμένο αριθμό σε διαγραφές που κυμαίνεται από 10.000 εώς 1.000.000 κύκλους σβησίματος.

Η υπέρβαση του αριθμού αυτού μπορεί να καταστήσει το μπλοκ ως μόνιμα αναξιόπιστο. Για παράδειγμα ένα multi-level cell (MLC) τύπου μπλοκ υποστηρίζει σχεδόν 10.000 κύκλους διαγραφής που σημαίνει πως αν διαγράφουμε ένα συγκεκριμένο μπλοκ συνέχεια και γράφουμε σε αυτό εκ νέου δεδομένα, τότε το όριο των κύκλων θα υπερβεί σε μόνο 3 ώρες. Εξαιτίας αυτού υπάρχουν κάποιοι μηχανισμοί εξισορρόπησης φθοράς (wear-leveling) που χρησιμοποιούνται από την μνήμη γι’ αυτόν τον συγκεκριμένο σκοπό ώστε να φθείρονται τα μπλοκ ομοιόμορφα.

Έτσι λόγω των παραπάνω τριών (3) χαρακτηριστικών της μνήμης flash η διαδικασία του σβησίματος/καθαρισμού από μη-έγκυρα μπλοκ (μπλοκ δηλαδή που περιέχουν μη-έγκυρες σελίδες) αναπτύχτηκε η ανάγκη εύρεσης μηχανισμών (αλγορίθμων)

καθαρισμού/συλλογής ‘’απορριμμάτων’’(garbage-collection algorithms), οι οποίοι θα μπορούν να δρουν σε όσο το δυνατόν μικρότερο χρονικό διάστημα, ελευθερώνοντας όσο περισσότερο χώρο μπορούν και χωρίς να διακόπτουν τυχόν λειτουργίες που βρίσκονται σε εξέλιξη.

3. Συλλογή απορριμμάτων (Garbage Collection)

3.1. FTL

Η μνήμη flash, προσπελαύνεται έμμεσα σαν μία συσκευή προσανατολισμένη σε μπλοκ (block-oriented device). Η αντιμετώπιση της μνήμης flash σαν μία συσκευή μπλοκ, επιτρέπει σε μπλοκ δεδομένων καθορισμένου μεγέθους, να εγγραφούν και να διαβαστούν όπως οι τομείς ενός δίσκου (disk sectors). Το FTL είναι ένα οδηγός διαχείρισης δεδομένων/αρχείων που δίνει την δυνατότητα σε μία μνήμη flash, όπως και σε κάθε συσκευή αποθήκευσης σε μπλοκ, να υποστηρίζει κάποια συστήματα αρχείων (file systems) όπως τα FAT, NTFS κάνοντάς την να συμπεριφέρεται σαν ένας μαγνητικός δίσκος. Τόσο ο οδηγός FTL όσο και ο οδηγός MTD (Memory Technology Device) είναι τα δύο κύρια στρώματα για την διαχείριση της μνήμης flash.

Ο οδηγός MTD παρέχει λειτουργίες χαμηλού επιπέδου: ανάγνωση, εγγραφή και διαγραφή. Έτσι αλγόριθμοι υψηλού επιπέδου, βασισμένοι στις παραπάνω λειτουργίες υλοποιούνται στον οδηγό FTL, αλγόριθμοι όπως: εξισορρόπησης φθοράς (wear leveling) , συλλογής ‘’απορριμμάτων’’(garbage-collection) και άλλοι.

3.2. Garbage-Collection και Wear-Leveling

Είναι ήδη γνωστό ότι η μνήμη flash είναι **μνήμη μονής εγγραφής**, και αυτό έχει ως αποτέλεσμα τα νέα δεδομένα να μην έχουν την δυνατότητα να εγγραφούν πάνω από τα παλιά (overwrite) σε κάθε διαδικασία ανανέωσης (update). Έτσι, τα νέα δεδομένα εγγράφονται μόνο σε ελεύθερο/διαθέσιμο χώρο και οι παλιές εκδόσεις δεδομένων κηρύσσονται ως μη έγκυρες (invalid).

Η παραπάνω στρατηγική καλείται ως **ανανέωση εκτός θέσης** (outplace update scheme), που σημαίνει πως μια σελίδα NAND μνήμης flash, πρέπει πρώτα να σβηστεί προτού τα νέα δεδομένα εγγραφούν σε αυτήν και στην ίδια ακριβώς θέση. Επίσης ένα άλλο χαρακτηριστικό της μνήμης flash είναι ο συγκεκριμένος αριθμός που μπορούν να σβηστούν τα μπλοκ και να ξαναεγγραφούν. Για παράδειγμα, κάθε

μπλοκ μιας τυπικής NAND μνήμης flash μπορεί να σβηστεί 1.000.000 φορές. Όταν ξεπεραστεί αυτό το όριο, το μπλοκ φθείρεται και γίνεται πιο ευάλωτο σε λάθη κατά την εγγραφή.

Αυτά τα συγκεκριμένα χαρακτηριστικά της μνήμης flash από μόνα τους καθώς και ο συνδυασμός τους δημιουργούν δύο πολύ σημαντικά προβλήματα στην μνήμη, με αποτέλεσμα να δημιουργηθούν διάφορες τεχνικές και αλγόριθμοι που προσπαθούν να τα λύσουν έχοντας ως σκοπό την αύξηση της διάρκειας ζωής της.

Μετά από ένα εύλογο χρονικό διάστημα χρήσης μιας μνήμης flash, ο ελεύθερος χώρος της μειώνεται αισθητά καθώς αποτελείται εκτός από τα δεδομένα του χρήστη και από πολλά μη έγκυρα δεδομένα. Αυτό έχει ως αποτέλεσμα την εκκίνηση δραστηριοτήτων που αποτελούνται από αναγνώσεις (reads), εγγραφές (writes) και σβησίματα (erases) με σκοπό την ανάκτηση ελεύθερου αποθηκευτικού χώρου. Οι δραστηριότητες αυτές συνθέτουν τον **μηχανισμό συλλογής απορριμμάτων (garbage collection mechanism)** και επιφέρουν έναν σημαντικό και μεγάλο φόρτο στην διαχείριση της μνήμης flash.

3.2.1. Μηχανισμός συλλογής απορριμμάτων (Garbage Collection)

Ο κύριος σκοπός του μηχανισμού συλλογής απορριμμάτων είναι να ανακυκλώνει τις σελίδες που περιέχουν απαρχαιωμένα και μη έγκυρα δεδομένα (dead and invalid pages) και να τις μετατρέπει σε ελεύθερες (free pages). Όμως κατά την διαδικασία του σβησίματος δεν μπορεί να επιλεχτεί μία σελίδα γιατί όπως γνωρίζουμε η διαδικασία αυτή γίνεται σε επίπεδο μπλοκ. Έτσι ένας άλλος σκοπός του μηχανισμού αυτού είναι η σωστή επιλογή των κατάλληλων μπλοκ προς ανακύκλωση, έτσι ώστε να ελαχιστοποιείται ο φόρτος που θα προκληθεί και ταυτόχρονα να υπάρξει η μεγαλύτερη δυνατή ανάκτηση ελεύθερου χώρου. Αυτός είναι και ο λόγος ύπαρξης αρκετών και διαφορετικών αλγορίθμων που προσπαθούν να βελτιστοποιήσουν τον μηχανισμό αυτόν, ορισμένους από τους οποίους θα περιγράψουμε και θα αναλύσουμε στο παρακάτω κεφάλαιο.

Για την δημιουργία ελευθέρου αποθηκευτικού χώρου που προορίζεται για τα νέα και ανανεωμένα δεδομένα θα πρέπει να ανακτηθούν τα απαρχαιωμένα μπλοκ που περιέχουν μη έγκυρες σελίδες. Η διαδικασία της ανάκτησης αυτού του ελεύθερου

αποθηκευτικού χώρου μπορεί να λάβει χώρα κατά το παρασκήνιο, όπου η μνήμη flash είναι αδρανής (idle), ή όταν το ποσοστό του ελεύθερου αποθηκευτικού χώρου πέσει κάτω από ένα συγκεκριμένο «κατώφλι» (συνήθως όταν πέσει κάτω από το 10% του ελεύθερου αποθηκευτικού χώρου).

Η ανάκτηση του ελεύθερου χώρου από τον μηχανισμό συλλογής απορριμμάτων πραγματοποιείται ακολουθώντας τα παρακάτω βήματα:

1ο Βήμα

Το πρώτο πρόβλημα που καλείται να λύσει ο κάθε αλγόριθμος είναι η εύρεση των μπλοκ που περιέχουν μη έγκυρες σελίδες και να επιλέξει με τα δικά του κριτήρια το πόσα και ποια θα είναι αυτά που πρέπει να σβηστούν.

2ο Βήμα

Αμέσως μετά την επιλογή των κατάλληλων μπλοκ προς αποθήκευση, επόμενος στόχος είναι η εύρεση έγκυρων σελίδων που τυχόν υπάρχουν μέσα στα μπλοκ και η άμεση αντιγραφή τους σε νέα ή σβησμένα (από προηγούμενη ενεργοποίηση του μηχανισμού) άδεια μπλοκ. Ο τρόπος με τον οποίο αντιγράφονται καθώς και σε ποια μπλοκ θα αντιγραφούν, αποτελούν και αυτά με την σειρά τους λόγους που διαφοροποιούν τους αλγορίθμους.

3ο Βήμα

Αφού αντιγραφούν όλες οι έγκυρες σελίδες σε νέα μπλοκ, το επόμενο και τελευταίο βήμα είναι το σβήσιμο όλων των επιλεγμένων μπλοκ και η επανατοποθέτησή τους στην λίστα ή λίστες με τα υπόλοιπα ελευθέρα μπλοκ, ώστε να επαναχρησιμοποιηθούν εκ νέου. Ο τρόπος με τον οποίο τοποθετούνται σε αυτές τις λίστες είναι σημαντικός από την άποψη φθοράς τους και αποτελεί τον τρόπο με τον οποίο αντιμετωπίζουν αυτό το πρόβλημα οι αλγόριθμοι συλλογής απορριμμάτων.

Τα 3 βασικά βήματα της διαδικασίας αυτής γεννούν κάποια ερωτήματα που οι απαντήσεις τους είναι αυτές που κάνουν τον κάθε αλγόριθμο μοναδικό και διαφορετικό από τους υπολοίπους. Για παράδειγμα ερωτήματα όπως:

1) Ποιες είναι οι προϋποθέσεις επιλογής των μπλοκ «θυμάτων» (victim blocks), των μπλοκ δηλαδή που θα επιλεγούν για σβήσιμο;

2) Πόσα μπλοκ πρέπει να επιλεχτούν;

3) Πώς ανακατανέμονται οι έγκυρες σελίδες;

4) Πώς χειρίζεται τα μπλοκ που ελευθερώνονται;

5) Πότε ή πόσο συχνά πρέπει να ενεργοποιείται ο μηχανισμός;

Τα ερωτήματα 1, 2 και 5 είναι αυτά που καθορίζουν το πώς λειτουργούν οι αλγόριθμοι συλλογής απορριμμάτων (garbage collection) ενώ τα 3, 4 και 5 καθορίζουν τη λειτουργία του μηχανισμού εξισορρόπησης φθοράς (wear leveling).

Καθώς παρατηρούμε τα παραπάνω βήματα, καταλήγουμε στο συμπέρασμα πως ένας αλγόριθμος συλλογής απορριμμάτων δεν αποτελείται μόνο από το την εύρεση και το σβήσιμο ορισμένων μπλοκ αλλά και από μια σειρά αναγνώσεων και εγγραφών σελίδων. Έτσι διαπιστώνουμε ότι για να είναι γρήγορος και αποτελεσματικός ένας αλγόριθμος δεν εξαρτάται μόνο από πόσα μπλοκ θα σβήσει συνολικά αλλά και ποια θα είναι αυτά τα μπλοκ που πρέπει να διαλέξει, καθώς μπορεί να περιέχουν πολλές σελίδες προς ανάγνωση και αντιγραφή. Σκοπός λοιπόν είναι να μελετηθούν εκείνα τα κριτήρια που ο συνδυασμός τους θα φέρει το επιθυμητό αποτέλεσμα.

3.2.2. Μηχανισμός Εξισορρόπησης Φθοράς (Wear Leveling)

Όπως φαίνεται οι περισσότεροι αλγόριθμοι συλλογής απορριμμάτων δίνουν μεγάλη σημασία, εκτός από τις βασικές λειτουργίες που πρέπει να υλοποιήσουν, και στο πως θα αντιμετωπίσουν με την σειρά τους την εξισορρόπηση φθοράς των μπλοκ, καθώς αποτελεί ένα μεγάλο μειονέκτημα αυτών των τύπων μνήμης flash και των δίσκων στερεάς κατάστασης (solid state disk).

Σε αυτή την εργασία αναλύονται αλγόριθμοι που είτε αδιαφορούν για την εξισορρόπηση φθοράς, είτε χρησιμοποιούν απλές τεχνικές είτε την τεχνική ανταλλαγής «καυτών-παγωμένων» δεδομένων (hot-cold data).

Κατά τη διαδικασία της συλλογής απορριμμάτων οι αλγόριθμοι οργανώνουν τα ελεύθερα μπλοκ σε μία ή και περισσότερες λίστες. Όσον αφορά τις απλές τεχνικές, που αναφέρθηκαν παραπάνω, χρησιμοποιούν μία λίστα από ελεύθερα μπλοκ, τα οποία είναι ταξινομημένα με την σειρά, ανάλογα με τον αριθμό σβησίματος που έχουν υποστεί. Έτσι τα μπλοκ με τον μικρότερο αριθμό σβησίματος (erase count)

βρίσκονται στην κορυφή της λίστας και είναι αυτά που θα επιλεχτούν αργότερα για μπλοκ προς εγγραφή. Με αυτό τον τρόπο τα μπλοκ με μεγάλο αριθμό σβησίματος θα καθυστερήσουν να επαναχρησιμοποιηθούν δίνοντας έτσι την δυνατότητα να μην φθείρονται γρήγορα.

Οι περισσότεροι αλγόριθμοι όμως χρησιμοποιούν την τεχνική ανταλλαγής «καυτών-παγωμένων δεδομένων» (hot-cold data) με αποτέλεσμα να χειρίζονται 2 λίστες ελεύθερων μπλοκ, την «καυτή» (hot) λίστα και την «παγωμένη»(cold) λίστα. Με τον όρο «καυτά» (hot) δεδομένα καλούμε τα δεδομένα που ανανεώνονται συχνά σε μικρό χρονικό διάστημα, ενώ «παγωμένα» θεωρούμε τα δεδομένα που ανανεώνονται σπάνια. Η διαφοροποίηση ανάμεσα στα δεδομένα γίνεται με διαφορετικά κριτήρια από κάθε αλγόριθμο, τα οποία θα αναλυθούν χωριστά παρακάτω, αλλά η έννοια παραμένει η ίδια.

Αντίστοιχα, και σύμφωνα με την πολιτική που ακολουθεί ο κάθε αλγόριθμος, η «καυτή» λίστα από ελεύθερα μπλοκ (Hot free list) περιέχει μπλοκ που ο αριθμός σβησίματός τους είναι μικρότερος από ένα «κατώφλι» (π.χ. από τον μέσο όρο των αριθμών σβησίματος όλων των μπλοκ), ενώ η «παγωμένη» λίστα (Cold free list) αποτελείται από μπλοκ με μεγαλύτερο αριθμό σβησίματος. Αυτές με την σειρά τους ταξινομούνται ανάλογα (όπως και στην απλή τεχνική) με αποτέλεσμα κατά την διαδικασία της ανακατανομής έγκυρων σελίδων να ομαδοποιούνται οι «καυτές» σελίδες σε μπλοκ από την «καυτή» λίστα και οι «παγωμένες» σελίδες σε «παγωμένα» μπλοκ αντίστοιχα. Σε γενικές γραμμές αυτή είναι η προσπάθεια που κάνουν από την μεριά τους οι αλγόριθμοι συλλογής απορριμμάτων για να μειώσουν σημαντικά την φθορά των μπλοκ και η οποία αναλύεται κατά την περιγραφή του κάθε αλγόριθμου ξεχωριστά.

3.3. Αλγόριθμοι συλλογής απορριμμάτων (Garbage Collection Algorithms)

Σύμφωνα με τα προηγούμενα κεφάλαια, στην μνήμη flash και στους δίσκους στερεάς κατάστασης δεν γίνεται να εγγραφούν νέα δεδομένα πάνω σε παλιά εκτός κι αν πρώτα αυτά σβηστούν. Επίσης, γνωρίζουμε ότι στην μνήμη flash υπάρχουν 3 βασικές λειτουργίες η ανάγνωση, η εγγραφή και το σβήσιμο που η καθεμία από αυτές χρειάζεται διαφορετικό χρόνο και ενέργεια για να πραγματοποιηθεί καθώς και ότι

λειτουργούν σε διαφορετικές μονάδες μεγέθους. Αυτές τις διαφορές μπορούμε να τις δούμε στην παρακάτω [Εικόνα 5].

Operation	Minimum execution time
Random page read	25 microseconds
Page program (write)	200 microseconds
Block erase	1500 microseconds

Εικόνα 5: Διαφορετικός χρόνος και μονάδες μεγέθους των λειτουργιών της μνήμης Flash (http://liobaashlyritchie.blogspot.gr/2013/01/the-nand-flash-architecture.html)

Έτσι λόγω του χαρακτηριστικού αυτού, δηλαδή ότι «πρώτα σβήνω μετά ξαναεγγράφω», υπάρχει ο μηχανισμός συλλογής απορριμμάτων που έχει σαν βασικό σκοπό «να καθαρίσει» την μνήμη από μη έγκυρα δεδομένα, δεδομένα δηλαδή τα οποία έχουν ανανεωθεί ή σβηστεί στην μνήμη. Επειδή όμως η λειτουργία του σβησίματος καταναλώνει τον περισσότερο χρόνο και ενέργεια από τις άλλες δύο(που και αυτές χρησιμοποιούνται κατά την διάρκεια του μηχανισμού) δημιουργήθηκε η ανάγκη ανάπτυξης έξυπνων αλγορίθμων οι οποίοι προσπαθούν, ο καθένας με τον δικό του τρόπο, να βελτιστοποιήσουν την λειτουργία του μηχανισμού συλλογής απορριμμάτων ως προς τον χρόνο και την ενέργεια. Για αυτό το λόγο άλλωστε παρατηρείται η αύξηση του αριθμού αυτών των αλγόριθμων μερικοί εκ των οποίων αναλύονται παρακάτω.

3.3.1. Αλγόριθμος Greedy

Το 1994 ο **Michael Whu** και ο **Willy Zwaenepoel** πρότειναν τον αλγόριθμο **Greedy**, ο οποίος υπήρξε ο πρώτος αλγόριθμος που υλοποιήθηκε για την συλλογή απορριμμάτων σε μνήμη flash και αποτέλεσε την βάση πάνω στην οποία στηρίχθηκαν οι επόμενοι καθώς και ως μέτρο σύγκρισης για την αποτελεσματικότητα τους ως προς αυτόν. Όπως είναι φυσικό ο αλγόριθμος **Greedy** είναι ο πιο απλός αλγόριθμος σε σχέση με τους μεταγενέστερούς του αλλά από τους πιο

αποτελεσματικούς στην εκκαθάριση της μνήμης έχοντας όμως αρκετά μειονεκτήματα.

Ξεκινώντας ο αλγόριθμος διαλέγει το μπλοκ με τις περισσότερες μη έγκυρες σελίδες. Για να το πετύχει υπολογίζει το utilization(***u***) όλων των μπλοκ και διαλέγει αυτό με το μικρότερο ***u***. Ο τύπος που υπολογίζει το u είναι:

$$\boldsymbol{u} = \frac{\#\ of\ valid\ data}{total\ \#\ of\ pages\ in\ a\ block}$$

Βρίσκοντας το μπλοκ με το μικρότερο ***u*** ελέγχει αν υπάρχουν έγκυρες σελίδες μέσα σε αυτό, και αν υπάρχουν, τις αντιγράφει στο επόμενο ελεύθερο/άδειο μπλοκ.

Αυτός ο «άπληστος» (greedy) τρόπος επιλογής του μπλοκ προς εκκαθάριση, αποτελεί και το μεγαλύτερο πλεονέκτημα του αλγόριθμου, πράγμα το οποίο αποδεικνύεται και από την χρήση αυτής της επιλογής και από άλλους αλγορίθμους όπως θα δούμε παρακάτω. Ο τρόπος αυτός αυξάνει αποτελεσματικά τον ελεύθερο χώρο αφού σβήνει μπλοκ με τις περισσότερες μη έγκυρες σελίδες που έχει ως συνέπεια να αντιγράφει και λιγότερες έγκυρες. Όσον αφορά τα μπλοκ, μετά το σβήσιμο τους δεν υπάρχει καμία ειδική μεταχείριση και τοποθετούνται απλά στο τέλος της λίστας των ελεύθερων μπλοκ. Όμως παρατηρείται ότι ο **Greedy** δεν ασχολείται καθόλου με την εξισορρόπηση φθοράς των μπλοκ, μειώνοντας έτσι την διάρκεια ζωής τους αισθητά.

3.3.2. Αλγόριθμος Cost Benefit (CB)

Ένα χρόνο αργότερα οι Atsuo Kawaguchi, Shingo Nishioka και Hiroshi Motoda προτείνουν έναν νέο αλγόριθμο για την συλλογή απορριμμάτων στη μνήμη, τον **Cost Benefit (CB)**. Σκοπός του είναι όχι μόνο η εκκαθάριση των μπλοκ με τις περισσότερες μη έγκυρες σελίδες αλλά και μπλοκ που περιέχουν τις πιο «παλιές» βοηθώντας με αυτό τον τρόπο την αντιμετώπιση της εξισορρόπησης φθοράς. Έτσι κάθε φορά ο αλγόριθμος υπολογίζει το Cost-Benefit (CB) του κάθε μπλοκ βάσει του παρακάτω τύπου:

$$\boldsymbol{CB} = \frac{(age * (1 - u))}{2 * u}$$

Όπου το age είναι ο χρόνος που έχει περάσει από την τελευταία τροποποίηση οποιασδήποτε σελίδας μέσα σε ένα μπλοκ. Δηλαδή η χρονική στιγμή που γίνεται μια σελίδα μη-έγκυρη. Ενώ το utilization($\boldsymbol{u}$) υπολογίζεται όπως και πριν:

$$\boldsymbol{u} = \frac{\#\ of\ valid\ data}{total\ \#\ of\ pages\ in\ a\ block}$$

Έχοντας υπολογίσει τα CB όλων των μπλοκ, διαλέγει αυτό με την μεγαλύτερη τιμή προς εκκαθάριση. Όσον αφορά την αντιγραφή των έγκυρων σελίδων σε άλλα μπλοκ όπως επίσης και την διαχείριση των ελεύθερων μπλοκ μετά το σβήσιμό τους, ο αλγόριθμος Cost-Benefit χρησιμοποιεί διαφορετικές μεθόδους από αυτές του Greedy, οι οποίες βοηθούν με την σειρά τους την εξισορρόπηση φθοράς. Οι μέθοδοι αυτοί στηρίζονται στην ύπαρξη 2 λιστών οι οποίες χωρίζονται σε «καυτές» και «παγωμένες» καθώς και στον διαχωρισμό των δεδομένων σε «καυτά» και «παγωμένα».

Ο διαχωρισμός των 2 λιστών γίνεται με βάση τον αριθμό σβησίματός τους. Δηλαδή αν ο αριθμός είναι μεγαλύτερος από το μέσο όρο των αριθμών σβησίματος όλων των μπλοκ τότε θεωρείται ως «παγωμένο», ενώ αν είναι μικρότερο ως «καυτό». Αυτό γίνεται για να αντιγράφονται τα «καυτά» δεδομένα (δηλαδή τα δεδομένα που ενημερώνονται πιο συχνά) στην «καυτή» λίστα που έχει μικρότερο αριθμό σβησίματος και αντίστοιχα τα «παγωμένα» δεδομένα στην «παγωμένη» λίστα. Έτσι δημιουργείται μια ισορροπία κάνοντας τα μπλοκ να φθείρονται όσο γίνεται ομοιόμορφα. Στο τέλος, μετά από το σβήσιμο του κάθε μπλοκ και αφού αποφασιστεί σε ποια από τις 2 λίστες θα μπει, σε κάθε λίστα γίνεται ταξινόμηση ως προς τον αριθμό σβησίματος του κάθε μπλοκ όπου στην κορυφή και των δύο βρίσκονται μπλοκ με τον μικρότερο αριθμό.

3.3.3. Αλγόριθμος Cost Age Time (CAT)

Το 1999 οι Mei-Ling Chiang, Paul C. H. Lee και Ruei-Chuan Chang δημοσίευσαν μία καινούρια πολιτική καθαρισμού, τον αλγόριθμο **Cost Age Time (CAT)**. Οι λόγοι οι οποίοι τον διαφοροποιούν σε σχέση με τους προηγούμενους είναι ότι προσπάθησαν να μειώσουν τον αριθμό σβησιμάτων που πραγματοποιούνται στα μπλοκ όπως επίσης και να αυξήσουν ακόμα περισσότερο την εξισορρόπηση της φθοράς στα μπλοκ. Για να το πετύχουν αυτό, στήριξαν την επιλογή του μπλοκ προς εκκαθάριση σύμφωνα με το κόστος καθαρισμού, το χρονικό διάστημα που βρίσκονται τα δεδομένα μέσα σε ένα μπλοκ και τον αριθμό που κάθε μπλοκ έχει υποστεί σβήσιμο, καταλήγοντας στον τύπο:

$$\boldsymbol{CAT} = \frac{u}{1 - u} * \frac{1}{age} * CT$$

Όπου age, ο χρόνος που έχει περάσει από την τελευταία ενημέρωση στο μπλοκ, CT ο αριθμός καθαρισμού του μπλοκ και u το utilization του κάθε μπλοκ το οποίο υπολογίζεται όπως και πριν:

$$\boldsymbol{u} = \frac{\#\ of\ valid\ data}{total\ \#\ of\ pages\ in\ a\ block}$$

Όταν έρθει η στιγμή της ενεργοποίησης του μηχανισμού για την εκκαθάριση των μπλοκ, ο αλγόριθμος αφού υπολογίσει τα CAT όλων των μπλοκ, επιλέγει εκείνο με την μικρότερη τιμή ως το μπλοκ «θύμα» (victim block) για να σβήσει το περιεχόμενό του. Η παραπάνω φόρμουλα δεν έχει ως στόχο να επιλέξει μόνο τα μπλοκ με τα περισσότερα μη έγκυρα δεδομένα αλλά όποτε είναι δυνατόν βρίσκει και μπλοκ τα οποία περιέχουν «πολύ παγωμένα» δεδομένα. Δηλαδή δεδομένα τα οποία δεν ανανεώνονται σχεδόν καθόλου με αποτέλεσμα το μπλοκ που τα περιέχει να μην έχει επιλεχθεί ποτέ προς εκκαθάριση με αποτέλεσμα να φθείρονται τα υπόλοιπα πολύ περισσότερο. Έτσι καταλαβαίνουμε πως κατά την εύρεση του μπλοκ «θύματος» ο αλγόριθμος δίνει μεγάλη σημασία και στην εξισορρόπηση φθοράς της μνήμης. Όσον

αφορά την αντιγραφή των έγκυρων δεδομένων αλλά και την διαχείριση των ελεύθερων/σβησμένων μπλοκ, ο αλγόριθμος ακολουθεί ακριβώς την ίδια λογική με τον Cost Benefit, δηλαδή την μέθοδο των 2 λιστών («καυτή», «παγωμένη») και τη διαφοροποίηση των δεδομένων σε «παγωμένα» και «καυτά».

3.3.4. Αλγόριθμος Endurant and Fast Greedy (EF-Greedy)

Το 2007 οι Ohhoon Kwon, Jaewoo Lee και Kern Koh παρουσιάζουν έναν νέο αλγόριθμο εκκαθάρισης απορριμμάτων σε μνήμη flash τον Endurant and Fast Greedy, που στην ουσία αποτελεί μία επέκταση του αλγόριθμου Greedy. Σκοπός τους είναι να προσπαθήσουν να μειώσουν τον αριθμό σβησίματος των μπλοκ, διότι γνωρίζουμε ότι αποτελεί την πιο αργή λειτουργία σε σχέση με την ανάγνωση και την εγγραφή, λαμβάνοντας υπόψιν την εξισορρόπηση φθοράς για την καλύτερη αντοχή της μνήμης στον χρόνο. Για την επιλογή των μπλοκ «θυμάτων» χρησιμοποιείται η πολιτική του Greedy η οποία διαλέγει τα μπλοκ με τις λιγότερες έγκυρες σελίδες που σημαίνει πως θα υλοποιηθούν λιγότερες αντιγραφές και θα ελευθερωθεί περισσότερος χώρος. Όμως λόγω του μειονεκτήματος αυτής της πολιτικής (δεν λαμβάνει υπόψιν την εξισορρόπηση φθοράς) οι δημιουργοί του χρησιμοποιούν μια καινούρια ιδέα για την ανακατανομή των έγκυρων σελίδων στα ελεύθερα μπλοκ. Κατά την νέα αυτή ιδέα διαχωρίζουν τις σελίδες και τα μπλοκ σε «καυτά» και «παγωμένα». Ο διαχωρισμός των σελίδων γίνεται βάση του διαστήματος ανανέωσης (update interval) της κάθε μίας, δηλαδή του πόσο συχνά ανανεώνεται κατά το παρελθόν. Για τον υπολογισμό του predicted inter-update time (PIU) της κάθε έγκυρης σελίδας στηρίζονται στην παρακάτω έκφραση:

$$\boldsymbol{PIU_k} = \frac{\sum_{j=k-n+1}^{k} I_j}{n}$$

Με το PIU προβλέπεται (predicted) το διάστημα ανάμεσα σε μία μελλοντική ανανέωση και στην τελευταία που έχει ήδη γίνει. Όπου *Ik* είναι ο (k)th inter-update time, *Ik+1* ο (k+1)th inter-update-time κοκ, και n πόσες φόρες έχει ανανεωθεί η σελίδα στο μπλοκ. Έτσι μελετάται η τελευταία ανανέωση αν n=1 ή όλες οι ανανεώσεις αν n=k, όμως η προκαθορισμένη τιμή του **n** δίνεται ίση με 3. Αφού έχουν υπολογιστεί τα PIU όλων των έγκυρων σελίδων μέσα στα μπλοκ προς εκκαθάριση,

αν η τιμή τους είναι μεγαλύτερη από το μέσο όρο όλων των PIU, τότε η σελίδα χαρακτηρίζεται ως «παγωμένη» και τοποθετείται σε «παγωμένο» ελεύθερο μπλοκ αλλιώς χαρακτηρίζεται ως «καυτή» και τοποθετείται σε «καυτό» ελεύθερο μπλοκ.

Τα μπλοκ με την σειρά τους διαχωρίζονται σε δύο λίστες ανάλογα με τον αριθμό σβησιμάτων που έχουν υποστεί. Αν ο αριθμός αυτός ξεπερνάει τον μέσο όρο όλων των σβησιμάτων όλων των μπλοκ τότε θεωρείται ως «παγωμένα» ενώ αν ο αριθμός είναι μικρότερος ως «καυτά». Τέλος αυτές οι δύο λίστες είναι ταξινομημένες ανάλογα πάλι με τον αριθμό σβησίματος, οπού στην κορυφή τους βρίσκονται αντίστοιχα τα μπλοκ με τον μικρότερο αριθμό.

3.3.5. Αλγόριθμος Swap Aware Garbage Collection (SAGC)

Το 2010 οι Ohhoon Kwon και Kern Koh δημοσιεύουν ακόμα έναν αλγόριθμο που αφορά την εκκαθάριση απορριμμάτων σε μνήμη flash, τον Swap Aware Garbage Collection (SAGC), ο οποίος δημιουργήθηκε κυρίως για μνήμες flash που χρησιμοποιούνται ως swap system. Αυτές οι μνήμες flash έχουν ως σκοπό να επεκτείνουν την κύρια μνήμη μιας ηλεκτρικής συσκευής λειτουργώντας όπως εκείνη (όταν η κύρια μνήμη δεν επαρκεί). Ο αλγόριθμος αυτός προσπαθεί να χρησιμοποιήσει για την επιλογή του μπλοκ «θύματος» μία πολιτική όμοια με αυτή του Greedy, γιατί η μέθοδος αυτή έχει ως αποτέλεσμα να απαιτούνται λιγότερες εγγραφές (καθώς διαλέγει μπλοκ με τα λιγότερα έγκυρα δεδομένα τα οποία θα μεταφερθούν) και κατά συνέπεια λιγότερο χρόνο για να τελειώσει η όλη διαδικασία. Έτσι για να το πετύχουν αυτό και ταυτόχρονα δίνοντας σημασία και στην εξισορρόπηση φθοράς κατέληξαν σε μία νέα πολιτική η οποία «σκέφτεται» πρώτον τον χρόνο που μια σελίδα είναι μη έγκυρη, δεύτερον τον swapped-out χρόνο, δηλαδή τον χρόνο που μια σελίδα αντιγράφεται από την κύρια μνήμη στην μνήμη flash και τρίτον τον αριθμό σβησίματος των μπλοκ.

Όταν ξεκινάει ο μηχανισμός εκκαθάρισης τότε ο αλγόριθμος υπολογίζει το Cost with Age Value (CAV) του κάθε μπλοκ και επιλέγει εκείνο με την μεγαλύτερη τιμή. Ο τύπος που υπολογίζει το CAV είναι:

$$\boldsymbol{CAV} = \sum_{i=0}^{n} i\_age \qquad \text{με}$$

$$\boldsymbol{i\_age} = c\_time - i\_time$$

Όπου **n** είναι ο αριθμός των μη έγκυρων σελίδων σε ένα μπλοκ, **c_time** είναι η τρέχουσα ώρα και **i_time** η ώρα όπου η κατάσταση της σελίδας άλλαξε από έγκυρη σε μη έγκυρη. Έτσι το **i_age** , είναι ο χρόνος που έχει περάσει από την ώρα που μια σελίδα έγινε μη έγκυρη.

Σύμφωνα με τον τύπο ο αλγόριθμος θα διαλέξει το μπλοκ με το μεγαλύτερο CAV. Αυτό σημαίνει από την μία ότι το μπλοκ περιέχει πολλές μη έγκυρες σελίδες και ότι έχουν προστεθεί τα **i_age** τους μεγαλώνοντας την τιμή κατά πολύ, ή από την άλλη ότι μπορεί το μπλοκ να περιέχει μόνο μια μη έγκυρη σελίδα η οποία να βρίσκεται σε αυτή την κατάσταση πολλή ώρα με αποτέλεσμα η τιμή του CAV να είναι μεγάλη λόγω του **i_time** της. Έτσι δεν επιλέγονται μόνο τα μπλοκ με τις περισσότερες μη έγκυρες σελίδες όπως στον Greedy αλλά και μπλοκ με απαρχαιωμένες σελίδες συμβάλλοντας έτσι στην καλυτέρευση της εξισορρόπησης φθοράς.

Όσον αφορά την αντιγραφή των έγκυρων σελίδων που υπάρχουν μέσα στο μπλοκ «θύμα» ο αλγόριθμος χρησιμοποιεί μία ακόμη μέθοδο για την ανακατανομή τους σε ελεύθερα μπλοκ. Πριν από την αντιγραφή τους ο αλγόριθμος υπολογίζει το Elapsed Swapped-out Time (EST), της κάθε μίας σελίδας, και ύστερα τις αντιγράφει στο ελεύθερο μπλοκ ταξινομημένες, δηλαδή πρώτα αυτές με το μεγαλύτερο EST και μετά αυτές με το μικρότερο. Αυτό γίνεται γιατί τα περισσότερα λειτουργικά συστήματα βασίζονται στην round-robin λογική με αποτέλεσμα η σελίδα που έγινε swapped-out πρώτη αυτή θα είναι πολύ πιθανό να γίνει και swapped-in στην κύρια μνήμη στο άμεσο μέλλον. Ο τύπος υπολογισμού του EST είναι:

$$\boldsymbol{EST} = c\_time - s\_time$$

Όπου c_time είναι η τρέχουσα ώρα και s_time η ώρα που μια σελίδα γίνεται swapped-out από την κύρια μνήμη.

Τέλος για την διαχείριση των ελεύθερων/σβησμένων μπλοκ η λογική είναι απλή. Χρησιμοποιείται μια λίστα με ελεύθερα μπλοκ η οποία είναι ταξινομημένη βάση του αριθμού σβησίματος του κάθε μπλοκ. Έτσι κατά την επιλογή του μπλοκ προς εγγραφή νέων δεδομένων ή την αντιγραφή των έγκυρων δεδομένων από μπλοκ που πρόκειται να σβηστούν, επιλέγεται πάντα το μπλοκ με το μικρότερο αριθμό σβησίματος.

3.3.5.1 Αλγόριθμος Swap Aware Garbage Collection (SAGC) με Long Endurance Policy (LEP)

Το 2012 οι Arushi Agarwal, Surabbi Maddhesiya, Priya Singh και Kumar Dwivedi πρότειναν τον αλγόριθμο SAGC με την Long Endurance Policy, μία πολιτική που τον κάνει να ξεχωρίζει από τον πρωτότυπο ως προς το πόσα μπλοκ «θύματα» επιλέγονται κάθε φορά. Κατά τον SAGC ο αριθμός αυτός δεν είναι συγκεκριμένος αφού κάθε φορά επιλέγονται random μπλοκ ενώ κατά την LEP πολιτική ο αριθμός αυτός προσαρμόζεται ανάλογα με το ποσοστό των ελεύθερων μπλοκ στην μνήμη. Ο αριθμός αυτός υπολογίζεται ως:

$$If\ n\_free < (n\_min - n\_free)$$

$$Then\ n\_victim = n\_free$$

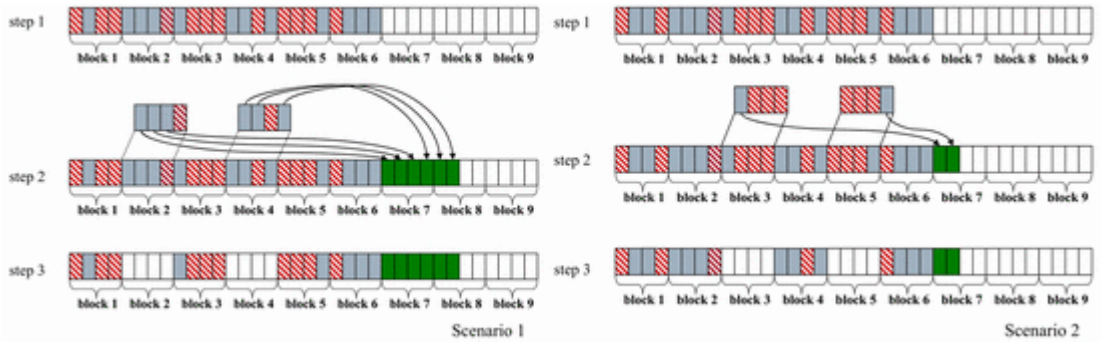
$$Else\ n\_victim = 2*(n\_min - n\_free)$$

Έτσι για παράδειγμα, όταν το ποσοστό των ελεύθερων μπλοκ πέσει κάτω από το 10% της συνολικής μνήμης τότε ξεκινάει ο μηχανισμός εκκαθάρισης και σταματάει όταν γίνει μεγαλύτερο από το 20%. Έστω ότι έχουμε 100 μπλοκ και ότι έχουν απομείνει 9 ελεύθερα (δηλαδή κάτω από το 10%), τότε σύμφωνα με την πολιτική θα επιλεγούν 9 μπλοκ προς εκκαθάριση (9 < (20 - 9)). Υποθέτουμε πως μετά την αντιγραφή των έγκυρων σελίδων και το σβήσιμο των μπλοκ έχουμε 15 μπλοκ ελεύθερα. Τότε πάλι σύμφωνα με την νέα πολιτική θα επιλεχτούν 10 (15 > (20 - 15)) μπλοκ κ.ο.κ. Το αποτέλεσμα είναι πως με αυτόν τον αλγόριθμο οι φορές που καλείται η διαδικασία της εκκαθάρισης είναι σε πολλές περιπτώσεις πολύ λιγότερες από ότι στον SAGC χωρίς την LEP πολιτική.

4. Υλοποίηση Αλγόριθμων σε Προγραμματιστική Γλώσσα Υψηλού Επιπέδου

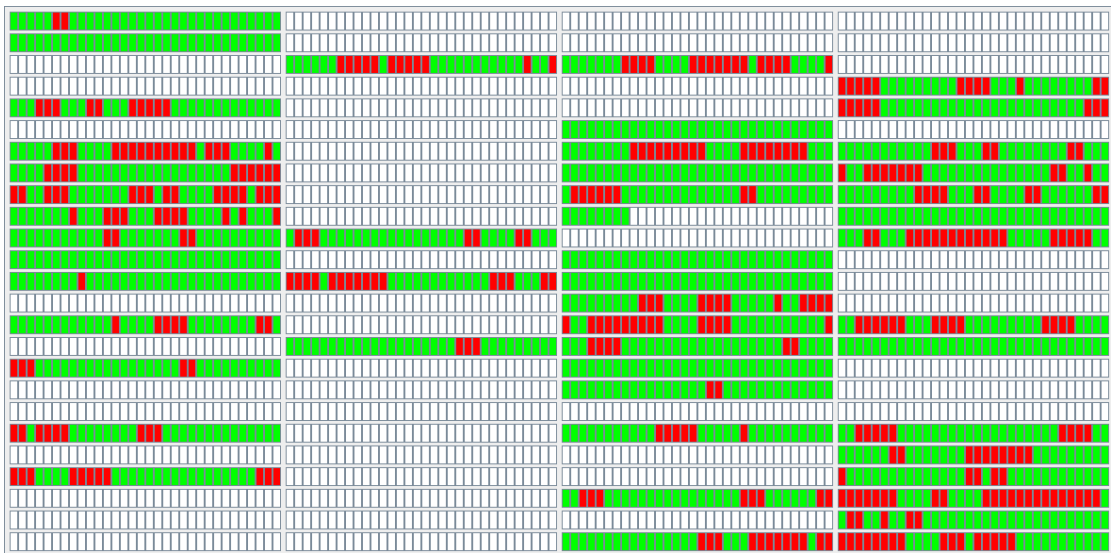
Σκοπός της παρούσας διπλωματικής είναι η υλοποίηση και η ανάλυση των παραπάνω αλγόριθμων σε μια προγραμματιστική γλώσσα υψηλού επιπέδου. Μετά από μία μικρή έρευνα στο διαδίκτυο καθώς και από τα συγκεκριμένα πειράματα που πραγματοποίησαν οι δημιουργοί των αλγόριθμων και που δημοσιεύουν στα αντίστοιχα επιστημονικά άρθρα, διαπιστώθηκε πως δεν έχει υλοποιηθεί κάτι παρόμοιο. Τα πειράματα αυτά πραγματοποιήθηκαν από επαγγελματικούς προσομοιωτές μνήμης flash διαφόρων μεγάλων εταιρειών όπως της Samsung, της HP και άλλων καθώς και από προσωπικούς προσομοιωτές τους οποίους απλά ανέφεραν χωρίς περισσότερες λεπτομέρειες. Έτσι μετά από μελέτη καταλήξαμε στην ιδέα της δημιουργίας ενός προγράμματος το οποίο θα υλοποιούσε μόνο την διαδικασία της συλλογής απορριμμάτων (Garbage Collection) βασισμένο στους παραπάνω αλγόριθμους. Η γλώσσα η οποία επιλέχθηκε είναι η JAVA λόγω των γραφικών της στοιχείων, τα οποία μας βοήθησαν να έχουμε μία οπτική επαφή με την λειτουργία των αλγόριθμων

Η αρχική ιδέα για το πώς θα δημιουργήσουμε το κατάλληλο πρόγραμμα ξεκίνησε από την μελέτη και την περιγραφή των αλγόριθμων στα αντίστοιχα επιστημονικά άρθρα, δηλαδή το πώς παρουσίαζαν και πως περιέγραφαν τις λειτουργίες της μνήμης flash. Για παράδειγμα η συνεχής χρήση της παρακάτω εικόνας



Εικόνα 6:Εικονική περιγραφή της αντιγραφής των έγκυρων σελίδων σε ελεύθερα μπλοκ κατά τη διαδικασία του σβησίματος
(http://www.citefactor.org/article/index/3090/pdf/a-long-endurance-policy-lep-an-improved-swap-aware-garbage-collection-for-nand-flash-memory-used-as-a-swap-space-in-electronic-devices)

η οποία δείχνει το πώς επιλέγονται τα μπλοκ προς εκκαθάριση και πως μεταφέρονται οι έγκυρες σελίδες στα ελεύθερα μπλοκ μας έκανε να σκεφτούμε, ποια θα είναι η διαμόρφωση της δικής μας εικονικής μνήμης flash η οποία φαίνεται στην παρακάτω εικόνα:



Εικόνα 7: Στιγμιότυπο γεμίσματος της εικονικής μνήμης Flash

Από την παραπάνω εικόνα παρατηρούμε πως η εικονική μνήμη flash στο πρόγραμμα αποτελείται από 100 μπλοκ τα οποία περιέχουν 32 σελίδες των 4KB η κάθε μία. Τα μπλοκ δημιουργούνται και αρχικοποιούνται από την κλάση **Blocks** η οποία περιέχει σχεδόν όλες τις βασικές λειτουργίες του κάθε αλγόριθμου. Έτσι κατά την εκκίνηση του προγράμματος δεσμεύονται 100 ελεύθερα μπλοκ (αποτελούνται μόνο από **λευκά** πλακίδια δηλαδή χωρίς δεδομένα) τα οποία γεμίζουν τυχαία ή και με συγκεκριμένη σειρά, ανάλογα τον αλγόριθμο, από εικονικά δεδομένα. Τα εικονικά δεδομένα δημιουργούνται και αυτά τυχαία από την κλάση **file** και είναι μεγέθους έως και 20KB και ο αριθμός τους δεν ξεπερνά τα 1000 (δηλαδή έχουμε 1000 μοναδικά αρχεία τα οποία επανεμφανίζονται τυχαία). Κατά την διάρκεια του γεμίσματος δεσμεύονται όσα μπλοκ χρειάζονται για την εγγραφή των νέων δεδομένων (πλακίδια με **πράσινο** χρώμα), ενώ αν βρεθεί αρχείο με το ίδιο όνομα (όχι κατά ανάγκη το ίδιο μέγεθος) θεωρείται ως ανανέωση και γίνεται έλεγχος για εύρεση παλιότερων εγγραφών ώστε να θεωρηθούν πλέον ως μη έγκυρα (πλακίδια με **κόκκινο** χρώμα). Η βασική λειτουργία του «γεμίσματος» ακολουθεί την παρακάτω διαδικασία:

```
Write()
{
    Allocate a free block;                                  (data placement)
    If new write
        Write data into the free block;
    else {
        /* perform the non-in-place-update */
        Mark the obsolete data as invalid;
        Write data into the free block;
     }
}
```

Όταν το μέγεθος των ελεύθερων μπλοκ γίνει μικρότερο από το 10% της συνολικής μνήμης (δηλαδή όταν έχουμε λιγότερα από 11 ελεύθερα μπλοκ) τότε ξεκινάει η διαδικασία του καθαρισμού ανάλογα με τον αντίστοιχο αλγόριθμο. Παρά τις διαφορές τους όλοι ακολουθούν την ίδια διαδικασία:

```
Cleaning()
{
    Select a victim segment for cleaning;                   (segment selection)
    Identify valid data in the victim segment;
    Copy out valid data to another clean flash memory spaces;   (data redistribution)
    Erase the victim segment;
    Enqueue the victim segment to free segment lists that are available for rewriting;
}
```

Όταν ελευθερωθούν αρκετά μπλοκ (μέχρι ο αριθμός των ελεύθερων μπλοκ είναι μεγαλύτερος του 20% της συνολικής μνήμης) τότε ξανά ξεκινάει το γέμισμα της μνήμης όπως ακριβώς περιγράψαμε παραπάνω.

Η βασική διαδικασία όλων των αλγόριθμων είναι αυτή που περιγράψαμε παραπάνω. Στο πρόγραμμά μας ο κάθε αλγόριθμος είναι υλοποιημένος από μία αντίστοιχη κλάση η οποία φέρει το όνομα του. Στην κάθε κλάση έχουν υλοποιηθεί όλες οι αντίστοιχες συναρτήσεις που χρειάζεται ο κάθε αλγόριθμος για την σωστή λειτουργία του, όπως για παράδειγμα συναρτήσεις υπολογισμού, επιλογής μπλοκ «θύματος», ανακατανομής έγκυρων σελίδων και διάφορες άλλες.

Η κλάση **Blocks** όπως έχουμε προαναφέρει περιέχει όλες τις πληροφορίες και τα δεδομένα που χρειάζεται ο κάθε αλγόριθμος. Περιέχει όλα τα δεδομένα των μπλοκ

και των σελίδων τους, όλα οργανωμένα στις αντίστοιχες λίστες και με τα κατάλληλα ονόματα. Επίσης περιέχει έναν constructor κατάλληλο για τον κάθε αλγόριθμο ξεχωριστά και συναρτήσεις οι οποίες καλούνται μέσω της κλάσης δίνοντας την δυνατότητα σε κάθε αλγόριθμο να καλεί μόνο ότι χρειάζεται. Για παράδειγμα υπάρχουν δύο συναρτήσεις δέσμευσης ελεύθερων μπλοκ για τα νέα δεδομένα ή για τα δεδομένα προς ανακατανομή διότι κάποιοι αλγόριθμοι χρησιμοποιούν δύο λίστες από ελεύθερα μπλοκ και κάποιοι άλλοι μία. Παρακάτω αναλύεται λεπτομερώς η κάθε κλάση και συνάρτηση που υλοποιήθηκε και χρησιμοποιείται από το πρόγραμμα:

4.1 Class Blocks (JButton, JLabel, Boolean, int)

Η κλάση Blocks καλείται στην αρχή κάθε αλγόριθμου για να γίνουν οι κατάλληλες αρχικοποιήσεις που είναι αναγκαίες για τον καθένα και η οποία περιέχει 4 ορίσματα όπου τα 3 πρώτα έχουν σχέση με το γραφικό περιβάλλον (το $3^{ο}$ έχει σχέση με την επιλογή αν θέλουμε ενεργοποιημένα τα γραφικά ή όχι) και το $4^{ο}$ είναι ο αριθμός του κάθε αλγόριθμου (0:Greedy, 1: EF_Greedy, 3:Cost_Benefit, 4: Cost_Age_Time, 5:SAGC, 6:LEP).

Συναρτήσεις:

int NUMofBlockswithInvalidPages(): Επιστρέφει τον αριθμό των μπλοκ που περιέχουν μη έγκυρες σελίδες.

int FreeBlockSize(): Επιστρέφει τον αριθμό των ελεύθερων μπλοκ.

int getDataofBlock(int, int): Παίρνει ως ορίσματα το μπλοκ και την σελίδα του και επιστρέφει τα όνομα των δεδομένων που περιέχει (τα ονόματα των δεδομένων είναι δοσμένα σε int).

int getEraseCount(int): Παίρνει ως όρισμα ένα μπλοκ και επιστρέφει τον αριθμό σβησιμάτων που έχει υποστεί.

void ChooseBlock(), void ChooseBlockEF(), void ChooseBlockSAGC(), void ChooseBlockEFGREEDY(String): Επιλέγουν το κατάλληλο ελεύθερο μπλοκ προς δέσμευση νέων δεδομένων ή δεδομένων προς ανακατανομή. Το όρισμα στην τελευταία συνάρτηση αποτελεί τον χαρακτηρισμό των δεδομένων σε “hot” ή “cold”,

η οποία συνάρτηση χρησιμοποιείται και από άλλους αλγορίθμους (παρά το όνομα της: **EFGREEDY**) για την ανακατανομή των σελίδων στα αντίστοιχα μπλοκ.

int Min_Index_Erase(): Επιστέφει το μπλοκ με το μικρότερο αριθμό σβησιμάτων από όλα τα μπλοκ.

void reserveEF(file), void reserveEF(int, int, String), void reserve(file), void reserveGREEDY(int, int), void reserveSAGC(file), void reserveSAGC(int, int): Οι συναρτήσεις με όρισμα File δεσμεύουν τα κατάλληλα μπλοκ που έχουν ήδη επιλεγεί για τα νέα δεδομένα, ενώ οι συναρτήσεις με τα διαφορετικά ορίσματα καλούνται για την δέσμευση των μπλοκ κατά την ανακατανομή των σελίδων στην διαδικασία της εκκαθάρισης. Είναι προφανές ότι ίδιες συναρτήσεις καλούνται από διαφορετικούς αλγορίθμους λόγω της ομοιότητας που παρουσιάζουν σε διάφορες λειτουργίες.

void search(file): Κατά την διαδικασία της εγγραφής νέων δεδομένων ψάχνει να βρει αν υπάρχουν παρόμοια δεδομένα που έχουν καταγραφεί στο παρελθόν ώστε να χαρακτηριστούν ως μη έγκυρα και να εγγραφούν με την σειρά τους τα «ανανεωμένα» δεδομένα.

void eraseGREEDY(int): Σβήνει όλο το μπλοκ από τις μη έγκυρες σελίδες καθώς αρχικοποιούνται και διάφορα δεδομένα που είναι χρήσιμα για τον κάθε αλγόριθμο. Η κλήση της φυσικά γίνεται αφού έχουν τερματιστεί οι προηγούμενες λειτουργίες της συλλογής απορριμμάτων. Τέλος τοποθετεί το μπλοκ στην κατάλληλη λίστα των ελεύθερων μπλοκ.

void Sort_HOTCOLD(): Ταξινομεί τις δύο αντίστοιχες λίστες έχοντας ως πρώτο στοιχείο το μπλοκ με το μικρότερο αριθμό σβησίματος.

Αυτές είναι οι βασικές συναρτήσεις της κλάσης Blocks οι οποίες χρησιμοποιούνται τόσο κατά την διαδικασία της συλλογής απορριμμάτων όσο και στην διαδικασία γεμίσματος της μνήμης flash με δεδομένα. Επίσης περιέχει και άλλες συναρτήσεις όπως για παράδειγμα οι **InitHC(), getAveragePIU(), getPIU(int, int), Update_I(file)** οι οποίες έχουν ως στόχο να παραδίδουν τις συγκεκριμένες πληροφορίες σε συγκεκριμένους αλγόριθμους τις οποίες καλούν.

Οι αλγόριθμοι είναι υλοποιημένοι από διαφορετικές κλάσεις που φέρουν το όνομά τους και περιέχουν κάποιες ομοιότητες μεταξύ τους όπως: η εγγραφή των δεδομένων

μέσα στην μνήμη και τα βασικά βήματα που ακολουθεί η διαδικασία της συλλογής απορριμμάτων. Από εκεί και πέρα η κάθε κλάση/αλγόριθμος περιέχει τις δικές της μοναδικές συναρτήσεις που την καθιστούν ως μοναδική. Αυτές τις συναρτήσεις περιγράφονται με λίγα λόγια παρακάτω.

***Σημείωση**: Δεν θα αναφερθούν οι τύποι ή οι εκφράσεις σύμφωνα με τις οποίες γίνονται οι υπολογισμοί αφού έχουν ήδη αναφερθεί στην ανάλυση των αλγόριθμων.

4.2. Class GREEDY()

void Calculate_Utilization(): Υπολογίζει το utilization του κάθε μπλοκ της μνήμης σύμφωνα με τον αλγόριθμο Greedy.

int Pick_Victim_Block(): Γίνεται η επιλογή του κατάλληλου μπλοκ προς εκκαθάριση (μπλοκ θύμα) σύμφωνα με τους υπολογισμούς που έχουν γίνει από την παραπάνω συνάρτηση.

void Copy_Valid_Pages_into_Free_Blocks(int): Αντιγράφονται οι έγκυρες σελίδες από το μπλοκ θύμα (όρισμα int) σε ελεύθερα μπλοκ.

4.3. Class EFGREEDY()

void Calculate_Utilization(): Υπολογίζει το utilization του κάθε μπλοκ της μνήμης σύμφωνα με τον αλγόριθμο Greedy.

int Pick_Victim_Block(): Γίνεται η επιλογή του κατάλληλου μπλοκ προς εκκαθάριση (μπλοκ θύμα) σύμφωνα με τους υπολογισμούς που έχουν γίνει από την παραπάνω συνάρτηση.

void Copy_Valid_Pages_into_Free_Blocks(int,double): Αντιγράφονται οι έγκυρες σελίδες από το μπλοκ θύμα (όρισμα int) σε ελεύθερα μπλοκ σύμφωνα με τον διαχωρισμό των σελίδων σε «καυτές» και «παγωμένες» κάτι το οποίο γίνεται από την σύγκριση του PIU της κάθε σελίδας με τον averagePIU (όρισμα double) όλων των σελίδων. Ο υπολογισμός του PIU και του averagePIU γίνονται με την κλήση των getPIU() και getAveragePIU() αντίστοιχα της κλάσης Blocks.

4.4. Class COST_BENEFIT()

void Calculate_Utilization(): Υπολογίζει το utilization του κάθε μπλοκ της μνήμης σύμφωνα με τον αλγόριθμο Greedy.

void Calculate_CB_Value(): Υπολογίζει την τιμή Cost Benefit της κάθε σελίδας των μπλοκ που περιέχουν μη έγκυρες σελίδες.

int Pick_Victim_Block(): Γίνεται η επιλογή του κατάλληλου μπλοκ προς εκκαθάριση (μπλοκ θύμα) σύμφωνα με τους υπολογισμούς που έχουν γίνει από την παραπάνω συνάρτηση.

void Copy_Valid_Pages_into_Free_Blocks(int,double): Αντιγράφονται οι έγκυρες σελίδες από το μπλοκ θύμα(όρισμα int) σε ελεύθερα μπλοκ σύμφωνα με τον διαχωρισμό των σελίδων σε «καυτές» και «παγωμένες» κάτι το οποίο γίνεται από την σύγκριση του αριθμού εμφάνισης των ίδιων δεδομένων σε κάθε σελίδα με το μέσο όρο (όρισμα double) όλων των δεδομένων.

4.5. Class COST_AGE_TIME()

void Calculate_Utilization(): Υπολογίζει το utilization του κάθε μπλοκ της μνήμης σύμφωνα με τον αλγόριθμο Greedy.

void Calculate_CAT_Value(): Υπολογίζει την τιμή Cost Age Time της κάθε σελίδας των μπλοκ που περιέχουν μη έγκυρες σελίδες.

int Pick_Victim_Block(): Γίνεται η επιλογή του κατάλληλου μπλοκ προς εκκαθάριση (μπλοκ θύμα) σύμφωνα με τους υπολογισμούς που έχουν γίνει από την παραπάνω συνάρτηση.

void Copy_Valid_Pages_into_Free_Blocks(int,double): Αντιγράφονται οι έγκυρες σελίδες από το μπλοκ θύμα(όρισμα int) σε ελεύθερα μπλοκ σύμφωνα με τον διαχωρισμό των σελίδων σε «καυτές» και «παγωμένες» κάτι το οποίο γίνεται από την σύγκριση του αριθμού εμφάνισης των ίδιων δεδομένων σε κάθε σελίδα με το μέσο όρο (όρισμα double) όλων των δεδομένων.

4.6. Class SAGC() & Class LEP()

void Calculate_CAV(): Υπολογίζει το Cost with Age Value των μπλοκ που περιέχουν μη έγκυρες σελίδες σύμφωνα με τον αλγόριθμο SAGC.

int Pick_Victim_Block(): Γίνεται η επιλογή των κατάλληλων μπλοκ* προς εκκαθάριση (μπλοκ θύματα) σύμφωνα με τους υπολογισμούς που έχουν γίνει από την παραπάνω συνάρτηση.

Κατά τον αλγόριθμο SAGC ο αριθμός των μπλοκ προς εκκαθάριση αναφέρεται και υλοποιήθηκε ως random, ενώ ο αριθμός των μπλοκ στην κλάση LEP γίνεται βάση υπολογισμού ο οποίος γίνεται από την συνάρτηση* ***int Calculate_LEP()* *αποτελώντας φυσικά και την βασική διαφοροποίηση των δύο αλγόριθμων.*

void Copy_Valid_Pages_into_Free_Blocks(int): Αντιγράφονται οι έγκυρες σελίδες από το μπλοκ θύμα (όρισμα int) σε ελεύθερα μπλοκ.

4.7. Υπόλοιπες Classes

4.7.1. Class Write()

Δημιουργεί 2 αρχεία .txt που το καθένα περιέχει random αριθμούς οι οποίοι είναι χρήσιμοι για τα ονόματα και τα μεγέθη των εικονικών δεδομένων που χρησιμοποιούνται. Η επιλογή αυτή της δημιουργίας των .txt έγινε για να χρησιμοποιηθούν ίδια δεδομένα κατά την δοκιμή των αλγορίθμων ενώ τα ονόματά τους παραπέμπουν στον αριθμό των δεδομένων που χρησιμοποιηθήκαν. Π.χ το file200000.txt και το size200000.txt αντίστοιχα δηλώνουν πως περιέχουν 200.000 αριθμούς/ονόματα εικονικών δεδομένων χωρίς βέβαια να ξεπερνούν τα όρια που έχουν οριστεί (20KB max, 1000 μοναδικά ονόματα).

4.7.2. Class file(int, int)

Δημιουργεί τα εικονικά δεδομένα που είναι αναγκαία για το γέμισμα της εικονικής μνήμης flash παίρνοντας ως ορίσματα το όνομα (επιλέχτηκε integer αντί String για λόγους διευκόλυνσης και σύγκρισης) και το μέγεθος. Η κλάση file περιέχει και διάφορες συναρτήσεις όπως getSize(), setSize(), getName(), getData(), updateData() οι οποίες είναι χρήσιμες κατά το τρέξιμο του προγράμματος.

4.7.3. Class flash_mem_sim()

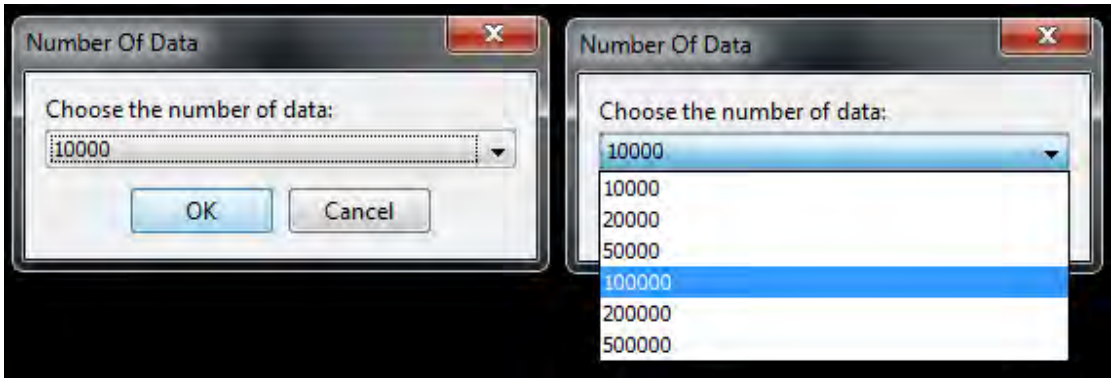
Η κλάση που δημιουργεί το γραφικό περιβάλλον το οποίο και βοήθησε στο να υπάρχει μία οπτική επαφή με την σωστή λειτουργία της εικονικής μνήμης και των αντίστοιχων αλγορίθμων κατά την διαδικασία της συλλογής απορριμμάτων.

5. Γραφικό Περιβάλλον και Αποτελέσματα

5.1 Γραφικό περιβάλλον

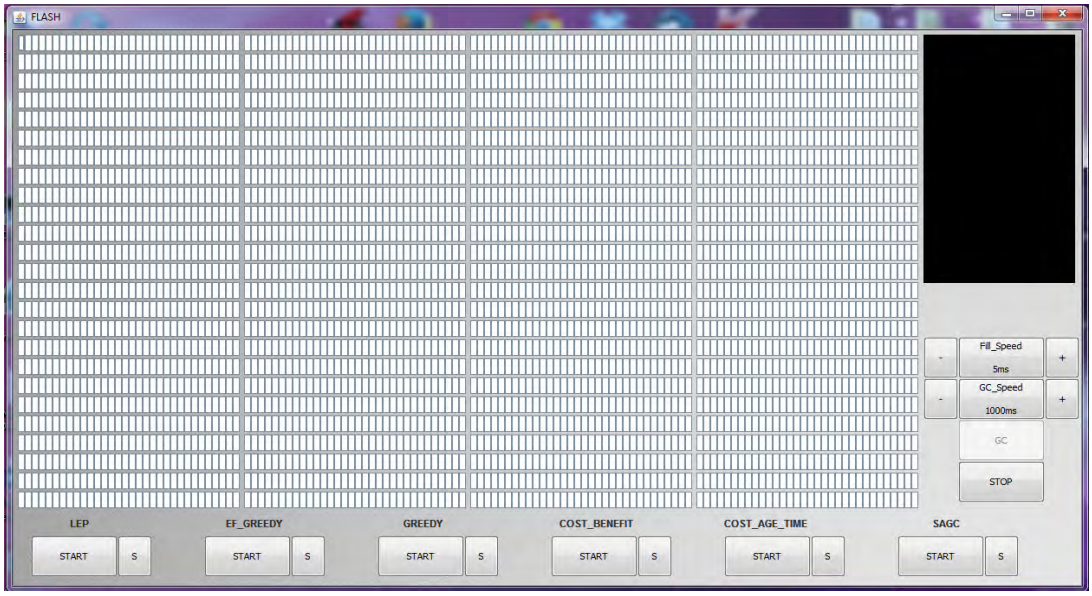
Όπως αναφέρεται στο προηγούμενο κεφάλαιο κατά την υλοποίηση της εργασίας χρησιμοποιήθηκε και το γραφικό περιβάλλον της Java ώστε να υπάρχει μία οπτική επαφή κατά το «τρέξιμο» των αλγόριθμων. Αν και στην αρχή χρησίμευσε μόνο για τον έλεγχο της ορθής λειτουργίας των αλγόριθμων στη συνέχεια αναπτύχθηκε περισσότερο, κάνοντας το κύριο μέρος της εργασίας, αφού ο χρήστης μπορεί να «τρέξει» τον κάθε αλγόριθμο μέσω αυτού πιο εύκολα και πιο γρήγορα, βλέποντας ταυτόχρονα την διαδικασία της εκκαθάρισης των δεδομένων καθώς και τα αποτελέσματα της.

Τρέχοντας λοιπόν το εκτελέσιμο αρχείο που έχει δημιουργηθεί με την εντολή "java –jar flash_mem_sim.jar" ανοίγει ένα παράθυρο [Εικόνα 8] το οποίο μας ζητά να διαλέξουμε πόσα δεδομένα θέλουμε να φορτωθούν στην εικονική μνήμη flash, έχοντας διάφορες επιλογές (10.000, 20.000, 50.000, 100.000, 200.000, 500.000 δεδομένα με μέγιστο μέγεθος 16KB).



Εικόνα 8: Παράθυρο επιλογής δεδομένων

Αφού επιλέξουμε τον αριθμό των δεδομένων και πατήσουμε «OK» θα ανοίξει με τη σειρά του το κύριο παράθυρο [Εικόνα 9] από το οποίο μπορούμε να διαχειριστούμε τον τρέξιμο του κάθε αλγόριθμου.

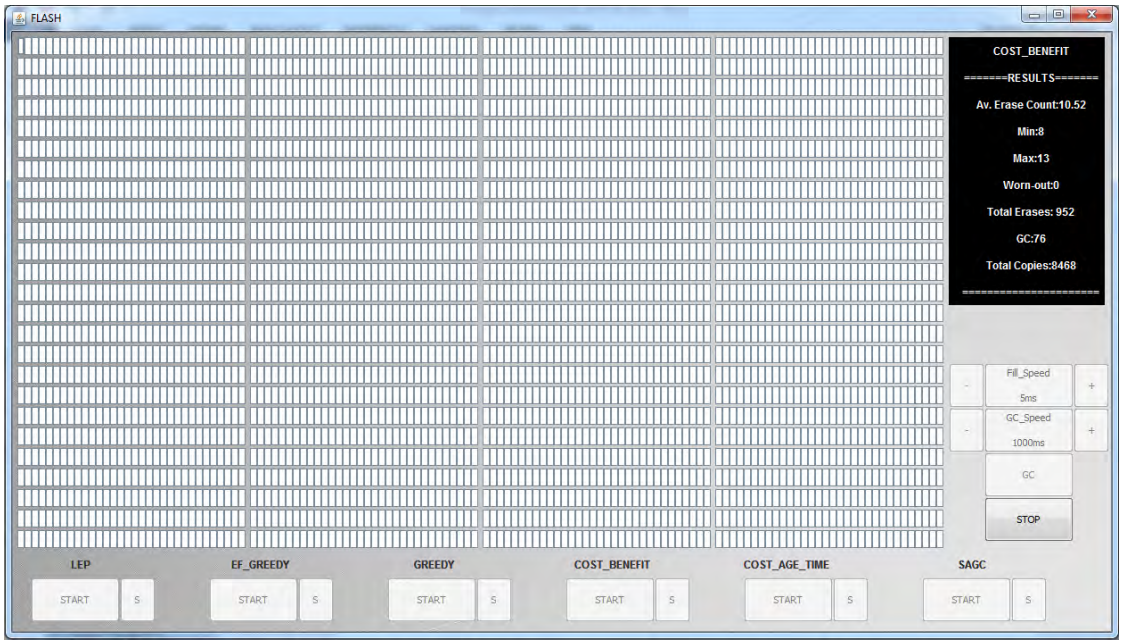


Εικόνα 9: Κύριο παράθυρο του προγράμματος

Όπως παρατηρούμε από την παραπάνω εικόνα βλέπουμε ότι υπάρχουν ορισμένα buttons τα οποία χρησιμεύουν για την εκκίνηση του κάθε αλγορίθμου καθώς και για την διακοπή του και την ταχύτητα με την οποία θα εκτελεστεί. Πιο συγκεκριμένα τα buttons “START” και “S” ξεκινάνε την διαδικασία γεμίσματος της μνήμης και όταν χρειαστεί (όταν η μνήμη φτάσει κάτω από το 10%) θα λειτουργήσει ο αντίστοιχος αλγόριθμος εκκαθάρισης ανάλογα με το αντίστοιχο button που έχει πατηθεί. Το button “S” κάνει ακριβώς την ίδια λειτουργία με το “START” μόνο που δεν ενεργοποιείται το «γραφικό γέμισμα» ώστε να τερματιστεί ο αλγόριθμος πολύ πιο γρήγορα (τα γραφικά όπως είναι φυσικό επιβαρύνουν την ταχύτητα εκτέλεσης). Τα buttons “Fill_speed”, “GC_Speed” και “GC” χρησιμεύουν για το «γραφικό γέμισμα» όπου τα 2 πρώτα, όπως μαρτυρεί και το όνομά τους, καθορίζουν την ταχύτητα με την οποία γεμίζει η μνήμη με δεδομένα και την ταχύτητα που εκτελείτε η διαδικασία του Garbage Collection αντίστοιχα. Το “GC” μπορεί να πατηθεί μόνο κατά τη διάρκεια του γεμίσματος αντιπροσωπεύοντας κατά κάποιο τρόπο το «idle time» κατά τη διάρκεια του οποίου ξεκινάει η διαδικασία εκκαθάρισης στις μνήμες flash. Το button “STOP” τερματίζει την κάθε διαδικασία σε όποιο στάδιο και αν βρίσκεται και εμφανίζεται στην θέση του το button “CLEAR” το οποίο καθαρίζει όλα τα δεδομένα φέρνοντας το παράθυρο στην κατάσταση που βρισκόταν όταν άνοιξε. Τέλος στο μαύρο πλαίσιο που υπάρχει εμφανίζονται η περιγραφή των βημάτων του Garbage

Collection όταν αυτό εκτελείται καθώς και τα αποτελέσματά του όταν ολοκληρωθεί η όλη διαδικασία ή τερματιστεί από τον χρήστη.

Εικόνα 10 : Cost_Benefit Start button



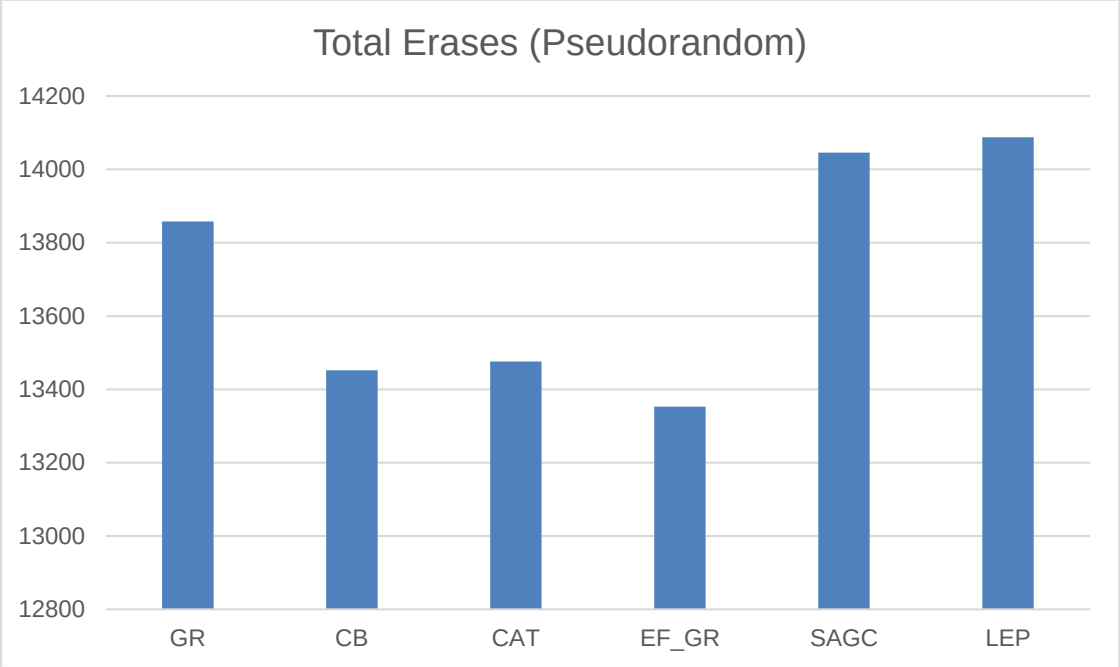
Εικόνα 11: Cost_Benefit S button

5.2 Αποτελέσματα

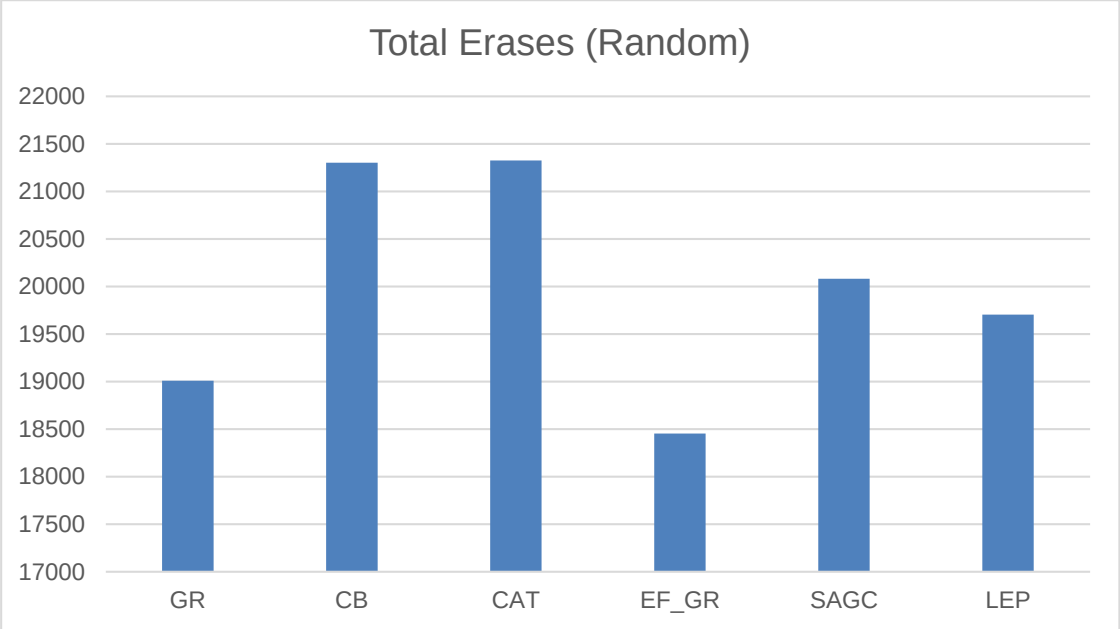
Σε αυτό το κεφάλαιο παρουσιάζονται τα αποτελέσματα των αλγορίθμων που υλοποιήθηκαν σύμφωνα με το πρόγραμμα το οποίο δημιουργήθηκε γι΄αυτό τον σκοπό.

Οι δοκιμές έγιναν κάνοντας χρήση 100.000 δεδομένων που δημιουργήθηκαν από την Random συνάρτηση της Java() και αποθηκεύτηκαν σε αρχεία txt ώστε να μην υπάρχουν διαφορετικές τιμές κατά την εκτέλεση του κάθε αλγόριθμου χωριστά. Τα δεδομένα είναι μεγέθους από 4KB έως 16KB και αποτελούνται από 1000 διαφορετικά (δηλαδή επαναλαμβάνονται 100.000 φορές 1000 διαφορετικά δεδομένα). Ο αριθμός των 1000 διαφορετικών δεδομένων προτιμήθηκε γιατί αν χρησιμοποιήσουμε μεγαλύτερο από αυτόν δεν θα υπάρχουν αρκετά μη έγκυρα δεδομένα να δημιουργηθούν και έτσι η μνήμη θα καθαριστεί γρήγορα και εύκολα χωρίς πολλές επαναλήψεις, ενώ αν διαλέγαμε μικρότερο από 1000 τότε θα είχαμε μπλοκ γεμάτα μόνο με μη έγκυρα δεδομένα χωρίς να είχαμε τις κατάλληλες αντιγραφές που χρειάζονται για την αξιολόγηση των αλγορίθμων. Τέλος χρησιμοποιούνται 100 μπλοκ των 32 σελίδων το καθένα (σελίδα των 4KB).

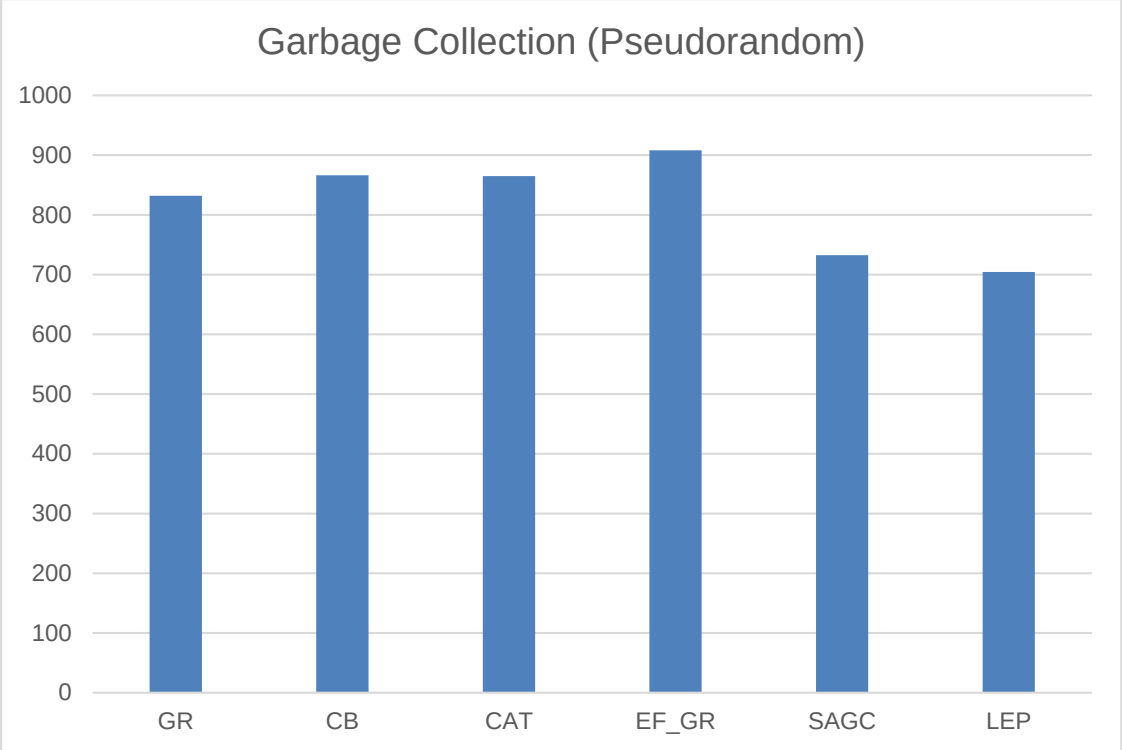
Τα αποτελέσματα που εμφανίζονται στα παρακάτω διαγράμματα αποτελούν τους μέσους όρους από 10 τρεξίματα του κάθε αλγόριθμου. Το Total Erases είναι τα συνολικά σβησίματα των μπλοκ που πραγματοποιήθηκαν, το Total Copies όλες οι αντιγραφές έγκυρων σελίδων από τα μπλοκ που επιλέχτηκαν για σβήσιμο και το Garbage Collection πόσες φορές ενεργοποιήθηκε ο μηχανισμός εκκαθάρισης. Το Blocks' Erase Counter παρουσιάζει τον μικρότερο, τον μεγαλύτερο και το μέσο όρο των μετρητών σβησίματος των μπλοκ μετά από 13000 σβησίματα ακριβώς. Τέλος πραγματοποιήθηκαν δύο δοκιμές, η μία με δεδομένα βγαλμένα από την Random συνάρτηση της Java και η δεύτερη με δεδομένα πάλι από την Random αλλά με κάποιες μικροεπεμβάσεις ώστε να έχουμε μια διαφορετική ανακατανομή των δεδομένων τα οποία από ότι θα παρατηρήσουμε μας δίνουν τελείως αντίθετο και ενδιαφέρον αποτέλεσμα σε σχέση με το πρώτο. Επομένως στα διαγράμματα αναφέρεται ως Random η πρώτη και ως Pseudorandom η δεύτερη.



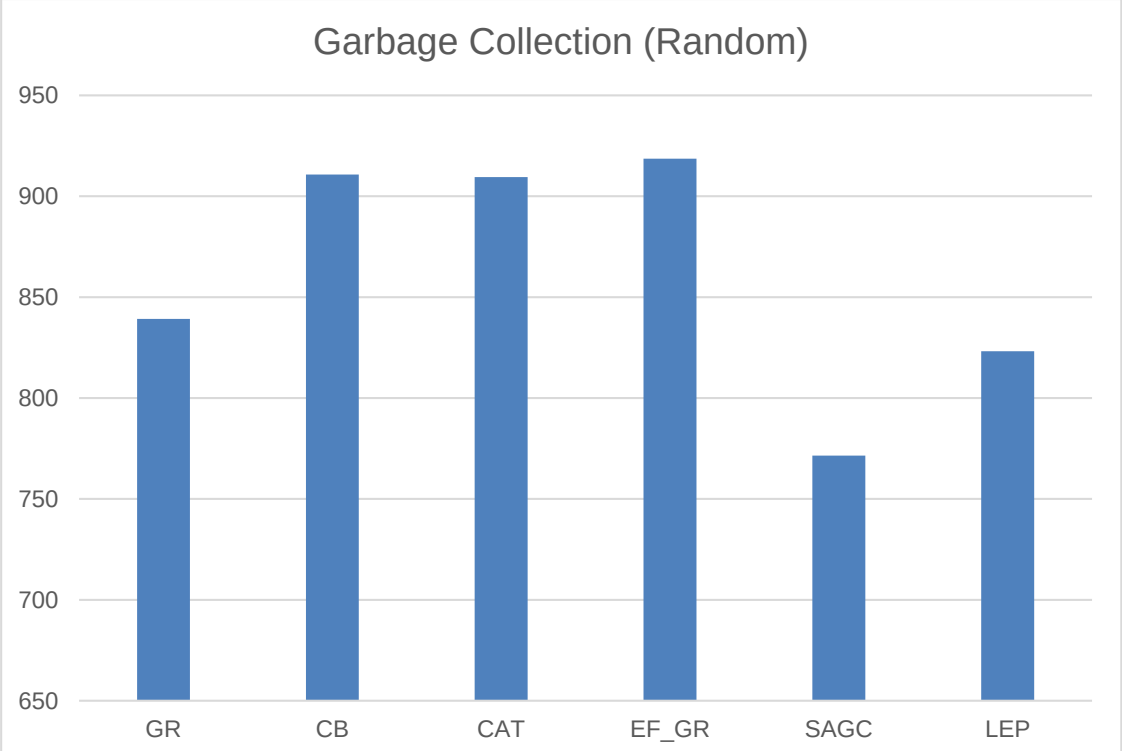
Διάγραμμα 1: Total Erases (Pseudorandom)



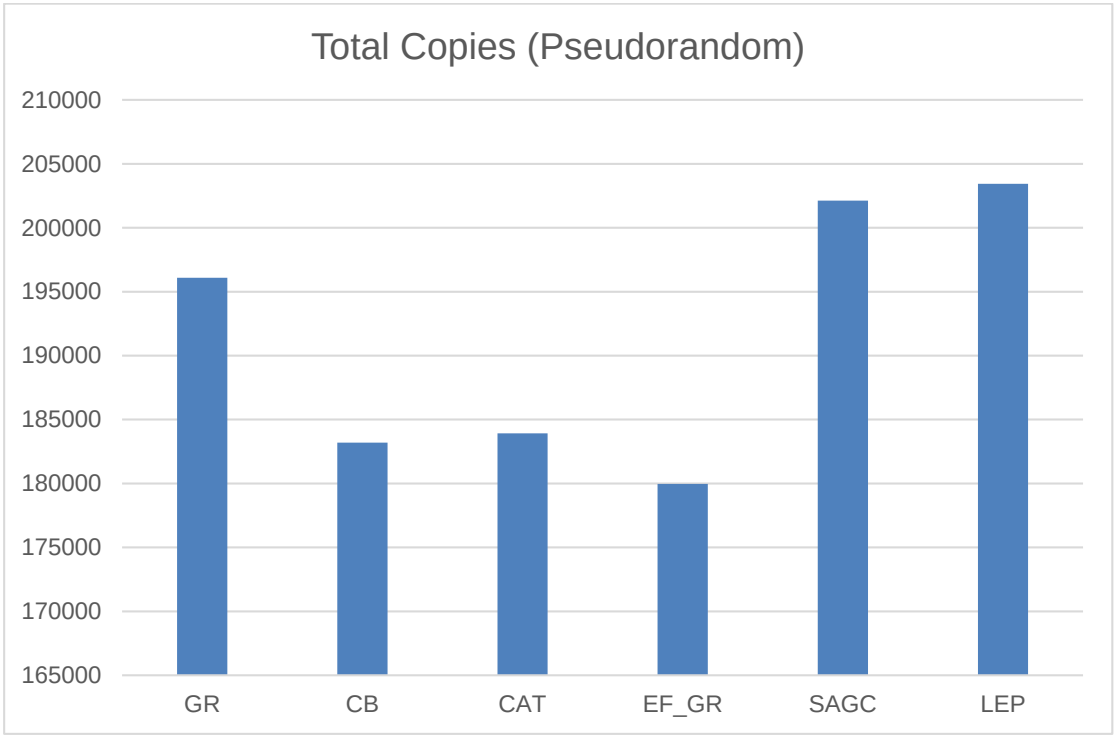
Διάγραμμα 2: Total Erases (Random)



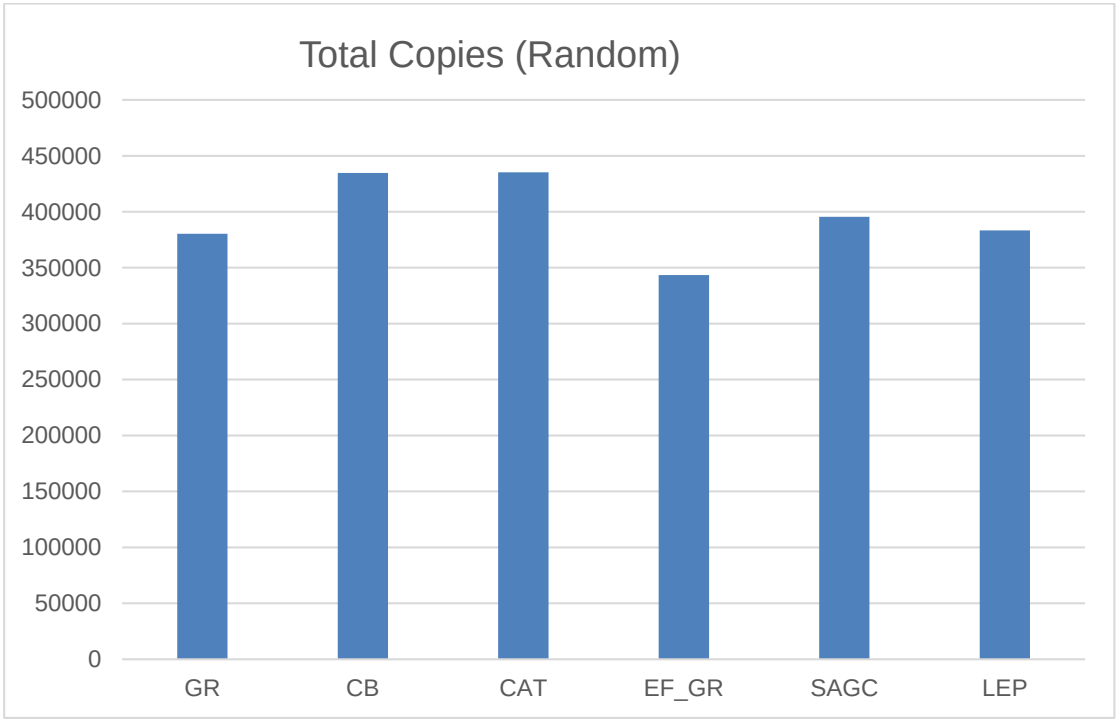
Διάγραμμα 3: Garbage Collection (Pseudorandom)



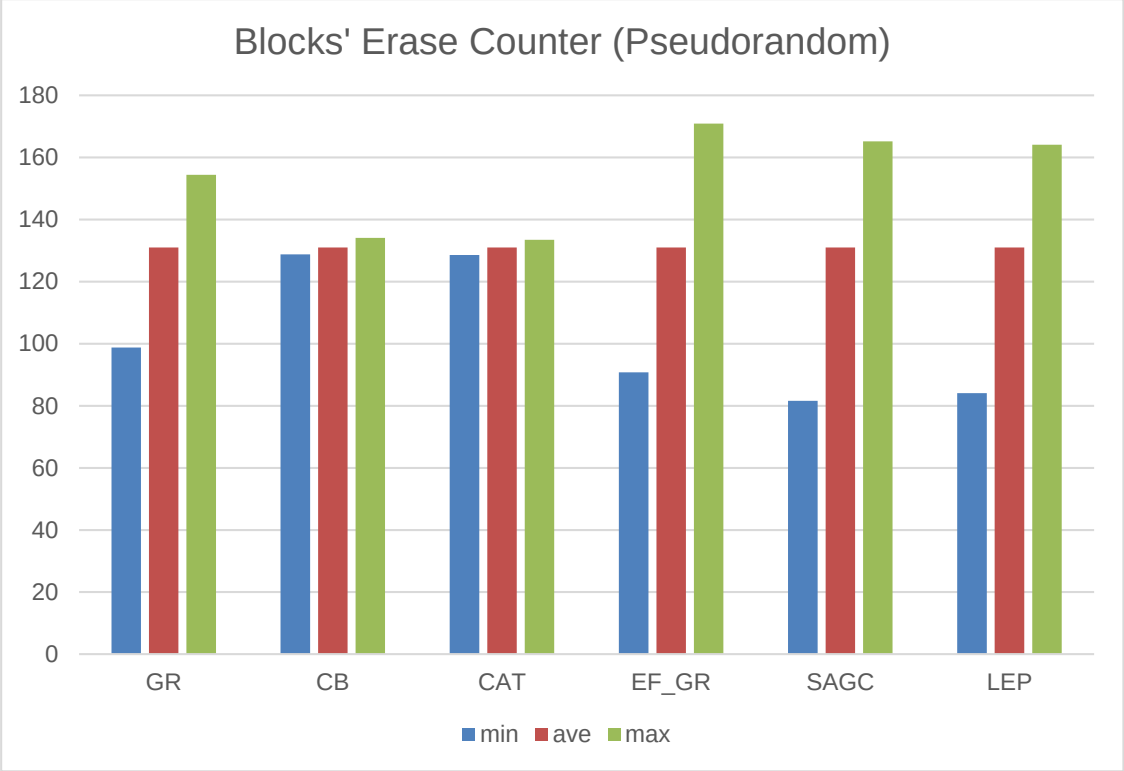
Διάγραμμα 4: Garbage Collection (Random)



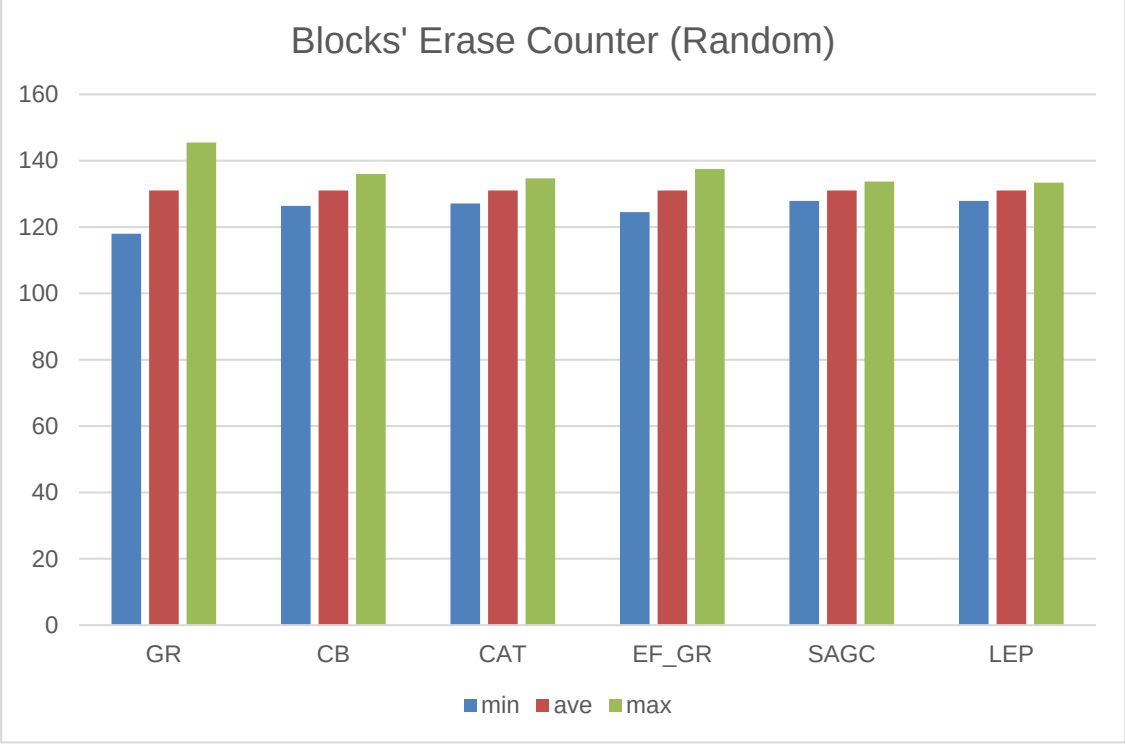
Διάγραμμα 5: Total Copies (Pseudorandom)



Διάγραμμα 6: Total Copies (Random)



Διάγραμμα 7: Blocks' Erase Counter (Pseudorandom)



Διάγραμμα 8: Blocks' Erase Counter (Random)

5.2.1 Παρατηρήσεις

Τα αποτελέσματα που βγήκαν από τις δοκιμές προέρχονται από το πρόγραμμα που δημιουργήθηκε για την υλοποίηση και την παρατήρηση των αλγόριθμων για Garbage Collection σε μνήμη Flash και Solid State δίσκους. Οι διαφορές που υπάρχουν ανάμεσα τους, όσον αφορά τα αποτελέσματα, δεν μπορούν να είναι εξίσου όμοιες με τις διαφορές που έχουν δημοσιευτεί στο κάθε επιστημονικό άρθρο στο οποίο αναλύεται ο κάθε αλγόριθμος. Αυτό συμβαίνει αφενός διότι οι δοκιμές που έχουν δημοσιευτεί έχουν γίνει σε επαγγελματικούς προσομοιωτές μνήμης flash και αφετέρου γιατί σε κάθε δοκιμή χρησιμοποιούνται μια σειρά από δεδομένα που κατά την διάρκεια του Garbage Collection καθιστούν ως καλύτερο τον αλγόριθμο τον οποίο παρουσιάζουν. Η παρατήρηση αυτή έγινε αφού διαπιστώθηκε πως ίδιοι αλγόριθμοι οι οποίοι χρησιμοποιούνται για σύγκριση έχουν πολύ διαφορετική συμπεριφορά στο καθένα από τα επιστημονικά άρθρα.

Επίσης η κάθε μία μνήμη flash έχει το δικό της τρόπο για το πώς ανακατανέμει τα δεδομένα σε σελίδες και κατά επέκταση σε μπλοκ καθώς και πως χειρίζεται το κάθε μπλοκ όσον αφορά την εξισορρόπηση φθοράς του. Αυτοί οι τρόποι φυσικά δεν αποκαλύπτονται από την κάθε κατασκευάστρια εταιρεία άλλα είναι σημαντικοί ως προς την αποτελεσματικότητα του Garbage Collection. Μία τελευταία παρατήρηση είναι πως οι περισσότεροι αλγόριθμοι (όχι όλοι) έχουν δημιουργηθεί για διαφορετική χρήση της μνήμης flash όπως για παράδειγμα ο Swap Aware Garbage Collection ο οποίος σύμφωνα με τους δημιουργούς του είναι κατάλληλος για μνήμες flash που χρησιμοποιούνται ως swap system.

Τέλος στα αποτελέσματα δεν μπορεί να συγκριθεί ο χρόνος του κάθε αλγορίθμου κατά την εκτέλεσή του διότι δεν θα είχε σχέση με την πραγματικότητα, αφού οι κώδικες τρέχουν σε Java και οι χρόνοι θα ήταν σχετικοί με το περιβάλλον της και όχι με μιας μνήμης flash.

Βιβλιογραφία

[1] Amir Rizaan Rahiman και Putra Sumari (2011) “Block Cleaning Process in Flash Memory”. Διαθέσιμο από:

http://www.intechopen.com/books/flash-memories/block-cleaning-process-in-flash-memory

[2] Arushi Agarwal et al. (2012) “A Long Endurance Policy (LEP): An Improved Swap Aware Garbage Collection For NAND Flash Memory Used As A Swap Space In Electronic Devices”, International Journal of Scientific & Engineering Research Volume 3, Issue 6, June – 2012. Διαθέσιμο από: http://www.citefactor.org/article/index/3090/pdf/a-long-endurance-policy-lep-an-improved-swap-aware-garbage-collection-for-nand-flash-memory-used-as-a-swap-space-in-electronic-devices#.U68m_0AXKSo

[3] Ohhoon Kwon και Kern Koh (2010) “Swap space management technique for portable consumer electronics with NAND flash memory”, Consumer Electronics, IEEE Transactions on Volume 56, Issue 3, Aug.- 2010. Διαθέσιμο από:

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5606292

[4] Ohhoon Kwon et al. (2007) “EF-Greedy: A Novel Garbage Collection Policy for Flash Memory Based Embedded Systems", Computational Science – ICCS 2007, 7th International Conference, Beijing, China, May 27 - 30, 2007, Proceedings, Part IV. Διαθέσιμο από:

http://link.springer.com/chapter/10.1007%2F978-3-540-72590-9_138

[5] Kawaguchi , S. Nishioka and H. Motoda (1995) "A Flash-Memory Based File System", *Proceedings of USENIX Technical Conference*, 1995. Διαθέσιμο από:

http://dl.acm.org/citation.cfm?id=1267424

[6] M.-L. Chiang , P. C. H. Lee and R.-C. Chang (1999) "Cleaning policies in mobile computers using flash memory", *Journal of Systems and Software*, vol. 48, 1999. Διαθέσιμο από:

http://www.sciencedirect.com/science/article/pii/S016412129900059X

[7] M. Wu and W. Zwaenepoel (1994) "eNVy: A Non-Volatile, Main Memory Storage System", *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994. Διαθέσιμο από:

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=348162&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D348162

[8] H. Kim , S. Lee and S. G. (1999) "A new flash memory management for flash storage system", *Proceedings of the Computer Software and Applications Conference*, 1999. Διαθέσιμο από:

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=812717&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D812717

[9] Longzhe Han,Yeonseung Ryu and Keunsoo Yim (2006) "CATA: A Garbage Collection Scheme for Flash Memory File Systems", Lecture Notes in Computer Science, Volume 4159, 2006. Διαθέσιμο από:

http://link.springer.com/chapter/10.1007%2F11833529_11

[10] Long-zhe Han,Yeonseung Ryu,Tae-sun Chung,Myungho and Lee,Sukwon Hong (2006) "An Intelligent Garbage Collection Algorithm for Flash Memory Storages", nternational Conference, Glasgow, UK, May 8-11, 2006. Proceedings, Part I. Διαθέσιμο από:

http://link.springer.com/chapter/10.1007%2F11751540_111

[11] http://liobaashlyritchie.blogspot.gr/2013/01/the-nand-flash-architecture.html

[12] http://whatis.techtarget.com/definition/NOR-flash-memory

[13] http://whatis.techtarget.com/definition/NAND-flash-memory

[14] http://embeddeddomain.blogspot.gr/2011/04/nor-vs-nand-flash.html

[15] http://en.wikipedia.org/wiki/Flash_memory

[16] http://en.wikipedia.org/wiki/Write_amplification

[17] http://en.wikipedia.org/wiki/Wear_leveling

[18] http://en.wikipedia.org/wiki/Flash_Translation_Layer

[19] http://codecapsule.com/2014/02/12/coding-for-ssds-part-3-pages-blocks-and-the-flash-translation-layer/